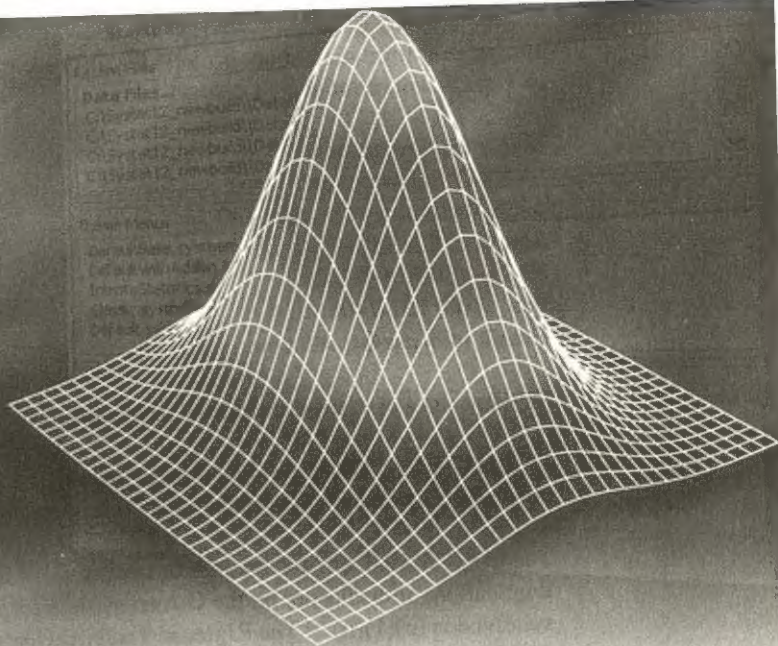


Data

~~Def Lib~~
FW Dec
SCERT Lib
28/5/11

SYSTAT
WWW.SYSTAT.COM

SYSTAT[®] 12



Data



SYSTAT[®]
WWW.SYSTAT.COM

For more information about SYSTAT® software products, please visit our WWW site at <http://www.systat.com> or contact

Marketing Department
SYSTAT Software, Inc.
1735 Technology Dr., Ste. 430
San Jose, CA 95110
Phone: (800) 797-7401
Fax: (800) 797-7406
Email: info-usa@systat.com

Windows is a registered trademark of Microsoft Corporation.

General notice: Other product names mentioned herein are used for identification purposes only and may be trademarks of their respective companies.

The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at 52.227-7013. Contractor/manufacturer is SYSTAT Software, Inc., 1735 Technology Drive, Suite 430, San Jose, CA 95110. USA.

SYSTAT® 12 DATA
Copyright © 2007 by SYSTAT Software, Inc.
SYSTAT Software, Inc.
1735 Technology Dr., Ste. 430
San Jose, CA 95110
All rights reserved.
Printed in the United States of America.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

1 2 3 4 5 6 7 8 9 0 05 04 03 02 01 00

8, 5, 2010
2996

Contents

List of Examples	ix
-------------------------	-----------

1 Working with Data	1
----------------------------	----------

2 Data Files	3
---------------------	----------

Data Editor	3
Opening Data Files	6
Import Formats.	7
Reading Variable Names	7
ASCII File Import	8
Excel, SigmaPlot and Lotus File Import.	8
dBASE File Import	9
SAS File Import	9
MINITAB File Import	10
STATISTICA File Import	10
JMP File Import	11
Import from Database with ODBC	11
Databases	12
Variable Selection	14
Saving Data Files	16
Save Options.	18
File Comments.	18
Export Formats.	19
Temporary Data Files.	20
Creating Temporary Data Files	21
Accessing Temporary Data Files	22
Merging and Appending Data Files	23

Append Variables	23
Append Cases	26
Using Commands	28
Examples.	29
References	33

3 Entering and Editing Data 35

Data Editor.	35
Data File Structure.	36
Variable Editor.	37
Pasting Variable Properties	40
Processing Conditions in Effect.	41
Variable Properties Dialog Box	42
To Specify Variable Properties	43
Data Entry	45
Editing Data.	47
Data Editor Navigation	47
Selecting Cells	48
Correcting Data Values	49
Changing Variable Names and Types	51
Adding Variable(s) or Case(s)	51
Dropping Variable(s)	53
Deleting Case(s).	53
Cutting and Pasting Variable(s) or Case(s)	54
Right Click Editing	56
Find Values in Column	59
Replace Values in Column	59
Find Variable	60
Go To	61
Undo/Redo	62
Case ID	63
Other Data Structures	63

Data in Tabulated Form.64
Matrix Input.66
Global Data Options.67
Using Commands69

4 Data Transformations **71**

Let Dialog Box73
If...Then Let Dialog Box74
Built-In Variables.76
Shortcuts for Transformations76
Operators and Functions.77
Arithmetic Operators78
Relational Operators78
Logical Operators.79
Order of Expression Evaluation81
Mathematical Functions82
Multivariable Functions84
Multicase Functions85
Group and Interval Functions86
Date and Time Functions86
Get the Day of the Week: DOW\$90
Character Functions90
Functions Relating to Probability Distributions96
Random Number Generators116
Rank Dialog Box117
Center Dialog Box.118
Standardize Dialog Box119
Trimming Data120
Reshaping Data121
Transpose121
Wrap/Unwrap.123
Stack124

Weight Dialog Box	125
The Calculator	126
Using Commands	127
Examples.	129
References	149

5 List, Sort, and Select **151**

List Cases Dialog Box	151
Picture Format	152
Additional Features for Listing Cases	153
Sort Dialog Box	154
Sort Order	155
Select Cases Dialog Box.	156
Using Commands	157
Examples.	158

6 Grouping Variables and By Groups **171**

Categorical Variables Dialog Box	172
NCAT Function	173
Value Labels Dialog Box	173
Category for Missing Codes	175
Order of Display Dialog Box	175
Recode Dialog Box	177
COD, INC and CUT Functions	179
Value Recoding: COD	179
Comparing Values: INC	180
Defining Intervals: CUT	180
By Groups Dialog Box.	180
By Groups versus Plot Groups	181

Using Commands	182
Examples	185

7 *Special Topics in Data Management* 193

Data Entry and BASIC	194
Free-Format Input	195
Fixed-Format Input	202
Troubleshooting ASCII Files	207
Export Data to an ASCII Text File	210
Select, List, and Save Cases	211
Programming Examples	215
Program Setup in BASIC	221
Statements and Expressions	221
IF...THEN Statement	221
ELSE Statement	222
FOR...NEXT Statement	224
WHILE...ENDWHILE Loop	225
FOR...NEXT Loops with Subscripted Variables	227
Dimensioning Space for New Variables	229
Subgroup Processing: BOG, EOG, BOF, EOF	232
PRINT Statement	234

8 *Matrix* 243

Matrix in SYSTAT	244
Read Matrix Dialog Box	245
Generate Matrix Dialog Box	247
Organize Dialog Box	249
Row/Column Operations Dialog Box	250
Matrix Operations Dialog Box	252
Decomposition Dialog Box	255

Using Commands	256
Examples	258
References	299

Acronym & Abbreviation Expansions 301

Index 311

List of Examples

Alphabetical Intervals	190
BEGINBLOCK...ENDBLOCK with ELSE Statement.	226
Calendar for the Year 2007-2008.	215
Categorizing Variables	190
Changing the Case of Character Values	135
Coding Missing Values.	199
Collapsing Categories	187
Computing F Statistic.	145
Computing Means of Subscripted Variables	230
Computing Means of Unsubscripted Variables: ARRAY	231
Computing the Area under a Curve	133
Computing Totals within Groups.	233
Concatenating Character Strings	141
Converting a Data File to a Text File.	210
Converting Numbers from European to American Notation	139

Covariance Matrices	206
Deleting Blanks or Other Characters	137
Deleting Selected Cases	212
Design Variables	279
Element-by-Element Operations and Functions	261
End-to-End Append	32
Entering Data and Defining Matrices	258
EOF: Printing the Last Case in a File	232
Extracting and Inserting Characters	138
Extracting First Names	140
Finding the Number of Missing Values	130
Fitting Normal Distribution with FOR...NEXT Statement	237
Fixed Format	204
Generating a New File of Random Data	143
Generating Uniform Random Numbers with WHILE ... ENDWHILE loop	235
Group Labels	160
ID Variables	159
IF statement with BEGINBLOCK...ENDBLOCK	226

IF...THEN DELETE:Using the first N Cases	236
IF...THEN...ELSE Statement	223
Inputting Data at the Command Prompt.	196
Inputting Fixed Format Data with Backslash (\).	205
Justifying Character Values	137
Labeling Multiple Variables.	187
Lagging Variables	129
Listing All Variables and Cases	158
Listing Cases Subsets for Selected Variables	158
Logging Ten Variables	228
Manipulating Matrices.	266
Matching Specific Values	134
Matching Values.	189
Matrices with Missing Diagonals: DIAGONAL	206
Matrix Algebra	270
Matrix Decomposition.	276
Merging with a Key Variable	31
Moore-Penrose Generalized Inverse	297

Multiple Regression and Canonical Correlation	295
Nested Sort	165
Omitting Variable Names and Case Numbers	213
Packing Two Records into One	281
Picture Format: Displaying Dates	164
Picture Format: Displaying Many Variables in One Panel	160
Printing the First Three Cases	214
Random Subsamples	236
Reading an ASCII Text File: GET	196
Reading Incomplete Records: Backslash (\)	202
Reading Multiple Cases per Record	201
Reading Multiple Lines per Case	197
Reading Records of Unequal Length	198
Recoding and Labeling Categories and Intervals	185
Recoding Characters to Numbers	189
Recoding Numeric Variables	188
Ridge Regression	286
Saving Data as Text without Quotes: PRINT	211

Saving Selected Cases to a New File	212
Selecting a Subset of the Variables	29
Selecting Cases with equal values of some Variables.	167
Selecting Subset of Cases	167
Side-By-Side Merge.	30
Simple Sort	165
Slope of the Line of Best Fit	131
Sorting in Ascending and Descending Order	166
Stacking Variables.	147
Standardizing Age.	144
Statistical Functions	272
Subgroup Means.	234
Summary Statistics	131
The Sweep Function.	288
Trimming Cases for Selected Variable(s).	146
Unpacking Records	209
Use of NCAT Function	191
Using PRINT to Save Selected Cases	213

Using Soundex to Code Names	142
Writing a Correlation Matrix as a Vector	282

Working with Data

Statistical and graphical analyses require data. Formatting data for a desired analysis can often be as challenging as interpreting the output. Fortunately, SYSTAT offers several transformations and manipulations designed to facilitate working with data. In addition, SYSTAT includes BASIC programming commands globally and includes matrix algebra, providing flexibility for advanced data management, under Utilities menu.

This manual focuses on tools available on the File menu, Edit menu, Data menu, and Utilities menu, and is organized as follows:

- Chapter 2 focuses on reading and writing data files.
- Chapter 3 describes the Data Editor. This editor serves both to view and to edit your data.
- Chapters 4, 5, and 6 introduce transformations and manipulations of data, including selecting cases, sorting cases, and stratifying analyses.
- Chapter 7 discusses the reading and writing of data in unique formats using the commands of the BASIC language.
- Chapter 8 describes SYSTAT's matrix algebra tools.

Data Files

Leland Wilkinson, Laszlo Engleman, Michael Pechnyo, and Lou Ross

A SYSTAT data file includes not only the data, but a great deal of information on the data files and the variables. These are explained in the sequel. You can create a new SYSTAT data file by entering data in the Data Editor or by importing data from another application.

Data Editor

The active file is displayed in the Data Editor, where you can also enter and edit data, run transformations, and select subsets of cases.

SYSTAT [C:\Program Files\Systat\Data\Ourworld.syc]

Stampage Untitled.syc Ourworld.syc

SYSTAT Output

C:\Program Files\Systat\Data\

COUNTRY	POP_1983	POP_1986	POP_1990	POP_2020	URBAN	BIRTH_82	BIRTH_91
1 Iceland	3 400	3 600	3 500	5 000	58 000	21 000	15 300
2 Austria	7 500	7 600	7 844	7 300	55 000	12 000	12 000
3 Belgium	10 200	9 900	9 909	9 500		12 000	12 000
4 Denmark	5 100	5 100	5 131	4 800	83 800	10 000	12 000
5 Finland	4 800	4 900	4 977	4 900	60 000	14 000	13 000
6 France	54 400	55 400	56 358	59 300	73 000	15 000	14 000
7 Greece	9 800	10 000	10 028	11 500	65 000	14 000	11 000
8 Switzerland	6 500	6 500	6 742	5 900	58 000	12 000	12 000
9 Spain	38 200	38 800	39 269	43 400	91 000	12 000	11 000
10 UK	58 300	58 500	57 364	56 600	76 000	12 000	14 000
11 Italy	57 500	57 200	57 884	54 800	69 000	11 000	10 000
12 Sweden	8 300	8 400	8 526	7 400	83 000	11 000	13 000
13 Portugal	10 100	10 100	10 354	12 100	30 000	16 000	12 000
14 Netherlands	14 400	14 500	14 938	13 800	88 000	12 000	13 000
15 West Germany	61 450	60 700	62 188	51 800	94 000	10 000	11 000
16 Norway	4 100	4 200	4 252	4 100	70 000	12 000	14 000
17 J. LNO	38 600	37 500	37 777	44 700	59 000	18 000	14 000

Output Examples Dynamic Data Variable

>USE Ourworld.syc

Interactive Log Untitled

POP_1983, POP_1986, POP_1990, POP_2020, URBAN, BIRTH_82, BIRTH_91

- To view the active data file in the Data Editor, from the menus choose:
View
Data Editor...
or
click on the Data tab of the data file in the Viewspace.
- To view the variable information of the variables used in the active data file you can click the Variable tab in the Viewspace
or
right-click on the Data tab in the Viewspace and select Data/Variable Editor.
- To open an empty Data Editor for creating a new data file, from the menus choose:
File
New
Data...
or
right-click the Data or Variable tab of the active data file in the Viewspace and select New from the context menu.

- To view multiple data files in the Data Editor, go to the Output Organizer, right click on the data file you want to view and select View Data. But you can use only one data file at a time.

Variable Tab. There is a Variable tab in Data Editor, besides the Data tab.

	VARIABLE NAME	VARIABLE LABEL	VALUE LABELS	TYPE	CATEGORICAL	CHARACTERS	DECIMALS	DISPLAY	DATE/TIME FO.	COMMENTS
1	COUNTRY	COUNTRY		String	NO	12				Names of the
2	POP_1982	POP_1982		Numeric	NO	12	3	Normal		Populations in
3	POP_1986	POP_1986		Numeric	NO	12	3	Normal		Populations in
4	POP_1990	POP_1990		Numeric	NO	12	3	Normal		Populations in
5	POP_2020	POP_2020		Numeric	NO	12	3	Normal		Populations in
6	URBAN	URBAN		Numeric	NO	12	3	Normal		Percentage of
7	BIRTH_82	BIRTH_82		Numeric	NO	12	3	Normal		Number of birt
8	BIRTH_RT	BIRTH_RT		Numeric	NO	12	3	Normal		Number of birt
9	DEATH_82	DEATH_82		Numeric	NO	12	3	Normal		Number of de
10	DEATH_RT	DEATH_RT		Numeric	NO	12	3	Normal		Number of de
11	BABYMT82	BABYMT82		Numeric	NO	12	3	Normal		Infant mortality
12	BABYMORT	BABYMORT		Numeric	NO	12	3	Normal		Infant mortality
13	LIFE_EXP	LIFE_EXP		Numeric	NO	12	3	Normal		
14	GDP_82	GDP_82		Numeric	NO	12	3	Normal		Gross nationa
15	GDP_86	GDP_86		Numeric	NO	12	3	Normal		Gross nationa
16	GDP_CAP	GDP_CAP		Numeric	NO	12	3	Normal		Gross domest
17	LOG_GDP	LOG_GDP		Numeric	NO	12	3	Normal		
18	ENR_82	ENR_82		Numeric	NO	12	3	Normal		Environmenta

Data Editor: Ourworld.sav

Interactive | Log | Untitled

The Data tab shows the data file. The Variable tab displays Variable Editor. There are two parts; one part provides information about variable names, variable type, width, display option, decimal places, variable comments, variable labels, value labels, and category variable status. The other part shows processing conditions like Frequency, Weight, Category, By, Case Selection, etc. Double-click on a variable name or click the Variable tab to get the Variable Editor.

You can see variable comments by moving the cursor to the variable name in the Data Editor. Labels as well as comments are displayed in the tool tip, in the form: labels: comments.

Variable Statistics. You can see summary statistics of a numeric variable, and a histogram, by right-clicking on the Variable name and choosing Variable Statistics.

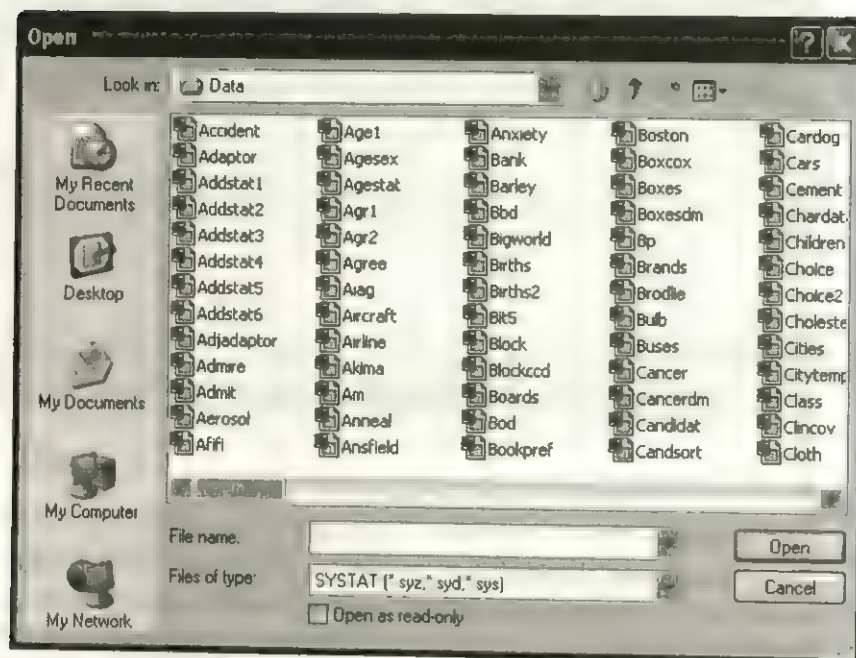
Opening Data Files

To open a data file, from the menus choose:

File

Open

Data...



SYSTAT data files are listed by default, and these files are read only. Descriptions of these data files are available in the Data Files Appendix of *Getting Started*. These are also available in the right-click of Data Editor tab in File Comments dialog and variable description in the Comments box of the Variable Properties dialog. To open a file of a different type, choose All Files from the Files of type drop-down list. The available types include: SPSS, SAS, Excel, SigmaPlot, Lotus, dBase, ASCII, MINITAB, STATISTICA, STATA, JMP, DIF, StatView and ArcView. To open a database of another type, use the ODBC import facility.

Import Formats

SYSTAT can open data files saved in the following formats:

- SYSTAT, FASTAT, and MYSTAT (*.SYS, *.SYD, *.SYZ)
- SIGMAPLOT (*.JNB)
- Excel (*.XLS)
- SPSS (*.SAV)
- SAS (*.SD2, *.SAS7BDAT, *.XPT, *.TPT)
- MINITAB (*.MTW)
- STATISTICA (*.STA)
- STATA (*.DTA)
- JMP (*.JMP)
- dBASE (*.DBF)
- ASCII text (*.TXT, *.DAT, *.CSV)
- ArcView (*.SHP)
- LOTUS (*.WK1, *.WK2, *.WKS)
- DIF files (*.DIF)
- STATVIEW (*.SVD)

To import of S-PLUS files you must employ SYSTAT's command language.

In addition, using ODBC import, SYSTAT can import data from any database with a corresponding driver installed on your computer.

Reading Variable Names

Variable name restrictions vary from product to product. To successfully import data from another application, variable names undergo any necessary modifications to yield valid, unique SYSTAT names. If the first line of data contains only labels (character strings), SYSTAT uses them as the SYSTAT variable names. If a variable name ends with a dollar sign (\$), SYSTAT reads the variable as a string variable, even if it contains numeric data. If a variable name does not end with \$, SYSTAT determines the variable's type from the first data value. SYSTAT appends a "V" to the beginning of any variable name beginning with an underscore. If there are no variable names in the

source file and if the first row contains numeric information, SYSTAT creates the variable names *VAR(1)*, *VAR(2)*, ..., *VAR(N)*.

Blank cells. SYSTAT considers blank cells to be numeric. Therefore, blank cells in the first row of a spreadsheet file are considered to be missing numeric values. This means that the first row is interpreted as data rather than as variable names.

Variable names. The length of the variable name can be up to 256 characters.

Subscripted variables. Variable names followed explicitly by subscripts (*i*), where *i* ranges from 1 to *n*, are subscripted when converted. String variable names can also be subscripted. For example- *RATS\$(1)*, *RATS\$(2)*.

ASCII File Import

An ASCII text file is created in a word processor or text editor and contains text and numbers with no special characters or formatting. Before you try to read an ASCII file, check the following:

- Each case is written as one line.
- The variable names are written at the top of the file and are separated by spaces and/or a comma.
- The values of the variables are separated by one or more spaces (blanks) and/or a comma.
- Missing numerical data are flagged by a period (.) and missing character data are marked by a space enclosed within quotation marks (" ").

Excel, SigmaPlot and Lotus File Import

The following general rules apply to reading of Excel, SigmaPlot and Lotus files:

- You can import Excel, SigmaPlot and Lotus worksheets only if they are saved in row and column order. Although Lotus can read files stored in other orders, row and column is the default.
- If an Excel file contains multiple worksheets, SYSTAT numbers the sheets in the order in which they appear in the file. You can access any sheet by specifying the sheet number. When you import an Excel file (*.XLS) that has multiple sheets, SYSTAT pops up a dialog box, that contains the names of the sheets with their sheet numbers. You can select the sheet you want from the drop down list. If a

SigmaPlot notebook (*.JNB) contains multiple worksheets, then only the first sheet will be imported.

- Any leading blank rows and columns are skipped.
- Numeric results of Lotus formulas are imported, with a warning issued to indicate that the imported data may not be accurate. If numbers were changed and the spreadsheet was saved before recalculating, the values saved may not reflect the changes that were made.
- Excel formulas cannot be imported; any cell containing an Excel formula appears as a missing value.
- Variables displayed in SigmaPlot using date/time formats are displayed in SYSTAT as numeric variables, which can be changed to date/time format by changing the variable properties.

dBASE File Import

The following general rules apply to reading of dBASE files:

- For dBASE III and IV, SYSTAT changes logical fields to character fields. The character used for the dBASE logical value is retained.
- SYSTAT skips memo fields, issuing a warning message when this occurs.
- Date fields are converted to numeric values. For example, if your dBASE file contained a value for November 2, 1987, it would be stored as 19871102. SYSTAT converts this to 0.19871102E + 8.

SAS File Import

SYSTAT reads data and transport files created by version 6 (release 6.08 or above) of SAS. However, the SAS procedure used to create a transport file determines SYSTAT's ability to read that file. Transport files created using PROC COPY, with the XPORT engine specified in the LIBNAME statement, can be imported. On the other hand, SAS transport files created using PROC CPORT cannot be imported. If a SAS transport file consists of multiple SAS data sets, SYSTAT gives a sheet number to each of the data sets. You can access the data sets by specifying the sheet number.

The following rules apply to the treatment of variables when importing SAS files:

Variable types.

- Variables displayed in SAS using numeric formats are displayed in SYSTAT as numeric variables.
- Variables displayed in SAS using string formats appear as string variables. SAS variables declared as strings are imported as string variables, *even if all values are numeric*.
- Variables displayed in SAS using date, time, or date/time formats are displayed in SYSTAT in a numeric format.

Variable values.

- Missing values in SAS files are supported in SYSTAT.
- SYSTAT does not import variable and value labels from SAS files.

MINITAB File Import

SYSTAT reads data created by versions 8-13 of MINITAB. Although MINITAB project files can contain multiple MINITAB worksheets along with other data, SYSTAT does not extract separate worksheets from these project files.

The following rules apply to the treatment of variables when importing MINITAB files:

- Variables displayed in MINITAB using numeric formats are displayed in SYSTAT as numeric variables.
- Variables displayed in MINITAB using date/time formats are displayed in SYSTAT as numeric variables, which can be changed to date/time format by changing the variable properties.
- Missing values in MINITAB files are supported in SYSTAT.

STATISTICA File Import

SYSTAT reads data created by version 5 of STATISTICA. The following rules apply to the treatment of variables when importing a STATISTICA file:

- Variables displayed in STATISTICA using numeric formats are displayed in SYSTAT as numeric variables.

- Variables having character values in STATISTICA are labeled and displayed as numeric values in SYSTAT.
- Variables displayed in STATISTICA using date/time formats are displayed in the SYSTAT numeric format. However, on changing the variable properties from numeric to date/time format, SYSTAT displays date/time with the chosen format.
- Missing values in STATISTICA files are supported in SYSTAT.

JMP File Import

SYSTAT reads data created by version 3.2 of JMP. The following rules are applied to the treatment of variables when importing a JMP file:

- Variables displayed in JMP using numeric formats are displayed in SYSTAT as numeric variables.
- Variables displayed in JMP using date/time formats are displayed in SYSTAT in numeric format. However, on changing the variable properties from numeric to date/time, SYSTAT displays date/time with the chosen format.
- Missing values in JMP files are supported in SYSTAT.

Import from Database with ODBC

Open Database Capture or Connectivity (ODBC) provides a method for reading data into SYSTAT from any database format for which a driver is installed on your system. ODBC import provides the ability to access data stored in formats which SYSTAT's standard import cannot recognize. In addition, you can also use ODBC import as an alternative to the standard import by using the appropriate driver. Because ODBC allows selection of variables and cases to import, you have greater control over the resulting data set.

ODBC import involves translating an import request from SYSTAT to the Structured Query Language (SQL). The ODBC driver translates the SQL to a format which a database type can process and returns the requested data. ODBC drivers included with SYSTAT allow importing data in the following forms:

- MS-Access
- Btrieve

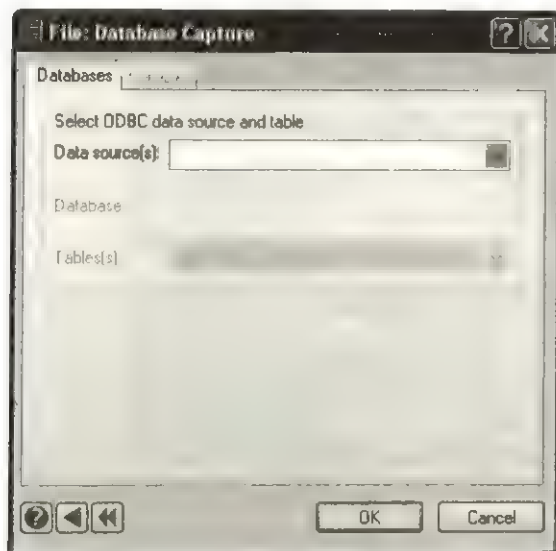
- DB2 Common Server (IBM)
- Clipper and FoxPro files
- INFORMIX
- INFORMIX 9
- OpenIngres
- OpenIngres 2
- Oracle
- Paradox tables
- PROGRESS
- SQL Server (Microsoft)
- SQLBase
- Sybase
- Text files

You define the data source, database, and variables to import using the Database Capture dialog box. This dialog box has two tabs: Databases and Variables.

Databases

To open the Database Capture dialog box, from the menus choose:

File
Database Capture...



Use the **Databases** tab to define the data source, select a database, and select a table in that database.

Data source(s). Click the box to view all data sources that have been set up for Windows using the ODBC Data Source Administrator. Select the source corresponding to the data to be imported.

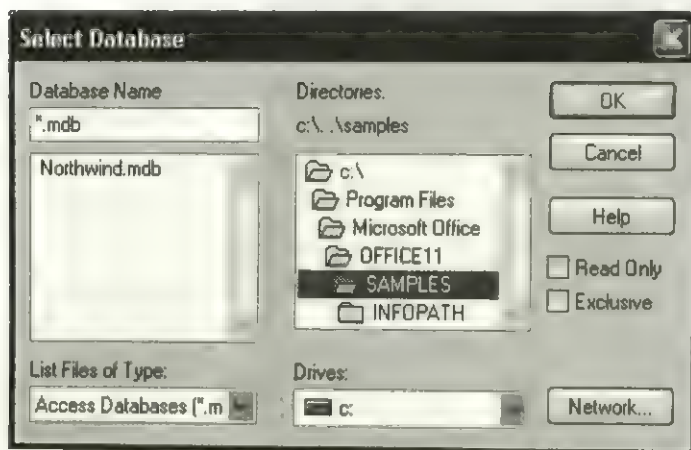
Database. If no database is associated with the selected data source, SYSTAT prompts you to select a database. After selecting a database, the database name and path appear on the **Databases** tab.

Tables. Click the **Tables** box to view all tables available in the selected database. Select the table to import into SYSTAT.

Clicking **OK** imports the entire selected table into SYSTAT. To import a subset of variables from the selected table, use the **Variables** tab.

Database Selection

If you have not set up the selected data source using the ODBC Data Source Administrator, the **Select Database** dialog prompts you for a database of the selected type to open.



The list of file types contains only the selected data source type and All Files. Selecting All Files allows you to view all of the files in the current directory, but you can only select a database that corresponds to the current data source. You can, however, select a file with a nonstandard extension as long as the file is consistent with the selected data source.

Passwords. If you select a database that requires a password, SYSTAT prompts you for the password. After successfully entering the password, you can access the database.

Variable Selection

Select the variables and cases to import using the Variables tab of the Database Capture dialog box.



Initially, ODBC selects all variables in the table for importing. Move variables to the Excluded variable(s) list to omit them from the imported data; only the variables in the Selected variable(s) list appear in the final data set.

The variable list consists of the variable names found in the database. During ODBC import, SYSTAT modifies these names as needed to yield valid variables in the resulting data set. Possible changes include:

- Removal of characters that are not letters, numbers, or underscores.
- Addition of a terminal dollar sign for string variables.
- Numbering of names that would otherwise be identical.

Use the SQL query area to submit custom database information requests, including statements to select cases.

Querying Databases Using SQL

Use the SQL query area of the Variables tab to create and submit custom database queries. SQL is the standard language used to access information in a database. A complete discussion of SQL is beyond the scope of this manual. To gain an understanding of SQL, we recommend Date and Darwen (1997), Melton and Simon (1993), and Bowman, Emerson, and Darnovsky (1996). Here, we will only discuss case selection and ordered imports using SQL.

Initially, the SQL Query area contains the word 'where'. The WHERE clause in SQL defines cases to import. Follow WHERE with conditions specifying values or ranges of values to import using =, <, >, <=, or >=. If a condition uses any character values, they must be enclosed in single quotation marks. Use AND or OR to combine conditions. For example, to import cases that have an AGE value below 30, specify

```
WHERE AGE<30
```

To restrict the import to males only, specify

```
WHERE AGE<30 AND GENDER='MALE'
```

Saving Data Files

To save a new data file, or to save an existing data file under a new name, from the menus choose:

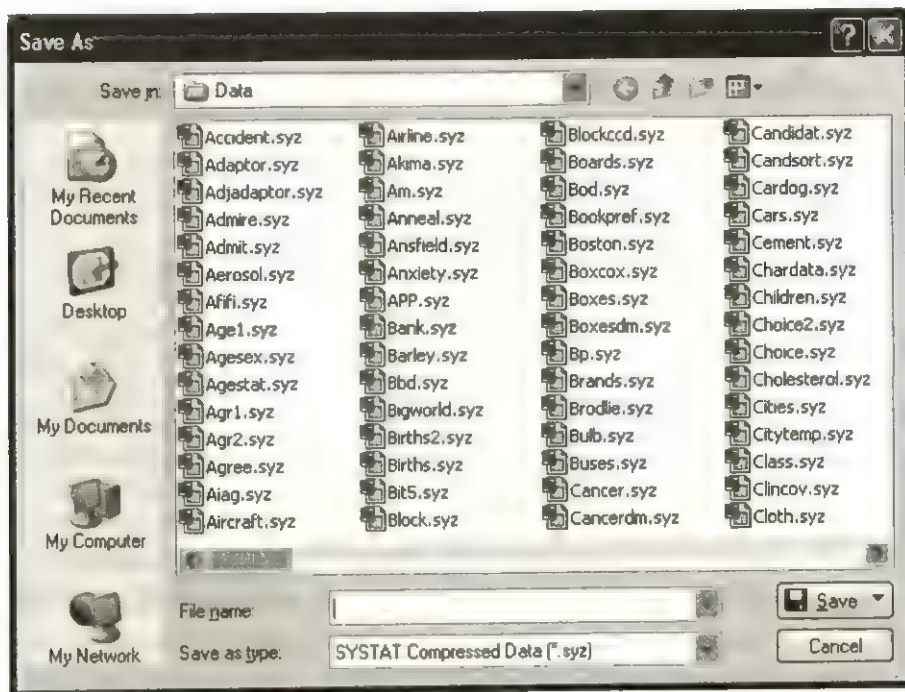
```
File  
  Save As...
```

or

```
File  
  Save  
    Data...
```

or

right-click the Data Editor tab for the active data file and select Save As from there.

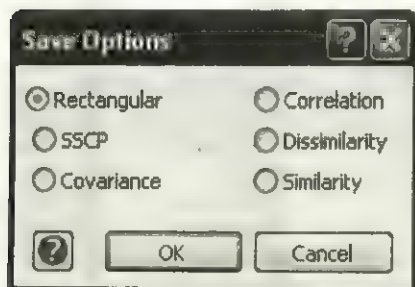


Specify a directory and filename. SYSTAT supports long filenames with a three-letter extension, and the names can contain letters or numbers. Use underscores for spaces. Filenames cannot include periods, aside from the period that separates the filename from the extension.

To save or export the data in a different format, such as a spreadsheet or database file, select the desired format from the Save as type drop-down list. When you save a file and do not specify an extension, SYSTAT automatically adds the default extension for that file type (for example, *.SYZ for SYSTAT data files). Data files are stored in a compressed format with the extension .SYZ along with variable and data information such as variable labels and value labels, and category information; because of this and also because the variable name length has increased, data file sizes are now larger than in earlier versions.

Save Options

When saving SYSTAT data files, you can click **Save Options** from the drop-down list of **Save** button, to specify a matrix type.



Available types include:

- **Rectangular.** Saves the raw data in a matrix of cases by variables.
- **SSCP.** Saves the data as a sums-of-squares and cross-products matrix.
- **Covariance.** Saves the data as a covariance matrix.
- **Correlation.** Saves the data as a correlation matrix.
- **Dissimilarity.** Saves the data as a dissimilarity matrix.
- **Similarity.** Saves the data as a similarity matrix.

File Comments

You can store a comment in your data file. SYSTAT displays the comment when you use the file in order to document your data files—for example, include the source of the data, the date they were entered, the particulars of the variables, etc. The comment can be as many lines as you want. If your comment is too long to fit on one line, use commas to continue onto subsequent lines. Enclose each line in single or double quotation marks:

```
DSAVE filename / 'Edited by Joe on Jan. 10',  
                  'Data from Dr. Jones'
```

Also you can right-click the **Data Editor** tab and select **File Comments** from it and then save the data file. You can also view this information by placing the cursor on the left topmost corner of the **Data Editor**.

To view the file comments in the output, employ the USE command with the COMMENT option:

```
USE filename / COMMENT
```

Export Formats

SYSTAT can export files in the following formats:

ASCII. Data are written in rectangular format, like a spreadsheet, in plain text rather than binary.

dBASE. Because dBASE requires fixed-format numeric fields, EXPORT first checks to find the best format to use for the numbers. The fixed format also limits the size of the values that dBASE accepts. If a number is larger than the field size allowed, SYSTAT writes the largest value that fits in that field. Missing values are inserted as blanks. Dollar signs (\$) are illegal characters in dBASE variable names and are replaced by underscores (_).

Excel and Lotus. SYSTAT writes missing values as blanks, string values without trailing blanks, and variable names exactly as they appear in the SYSTAT file.

Data Interchange Format (DIF). Missing values are written as SYSTAT missing values. String values are written without trailing blanks.

MINITAB. SYSTAT exports MINITAB data files (*.MTW) for Version 13 worksheets only. When string variables are exported to MINITAB, the column numbers are suffixed with '-T'.

STATISTICA. SYSTAT exports STATISTICA data files with (*.STA) extension. Statistica does not have a string type, so character variables cannot be exported. When writing a Statistica file, SYSTAT will use a value of -9999 for missing.

JMP. Dollar signs (\$) are removed from the string variables when exporting into JMP. Missing string values are replaced with a blank.

S-PLUS. SYSTAT exports S-PLUS data files through commands only. For missing values SYSTAT will write N.A.

STATA. SYSTAT exports STATA data files with (*.DTA) extension.

SAS. SYSTAT exports SAS data files for the Windows platform (*.SD2) and SAS transport files (*.XPT). Transport files exported from SYSTAT must be read in SAS

using PROC COPY (with the XPORT engine specified in the LIBNAME statement). Missing values are written as SAS missing values.

See *Language Reference* for the export command for each format.

Writing SAS Variables

When exporting data to a SAS format, variable names may be modified to satisfy SAS naming requirements. In version 6 of SAS, variable names have a maximum length of eight characters and can contain letters, numbers, and underscores. Furthermore, names must begin with a letter or an underscore. The following general rules are applied as needed to transform SYSTAT variable names to valid SAS names:

- SYSTAT removes any parentheses and \$ symbols.
- All characters beyond the eighth are dropped.
- If the resulting names are not unique, SYSTAT replaces the rightmost characters with digits incrementing from 001 until all names are unique.

Although the preceding rules yield variable names that vary between SYSTAT and SAS data files, the order of the variables in both files is identical.

In addition, SYSTAT variable types need to be exported into valid SAS variable types. The correspondence between variable types in the two applications is as follows:

- Numeric SYSTAT variables appear in SAS using one of SAS's numeric formats.
- String variables in SYSTAT are displayed in SAS as strings.
- Variables using date formats in SYSTAT use the MMDDYY format in SAS.
- Variables using time formats in SYSTAT use the TIME format in SAS.
- Variables using date/time formats in SYSTAT use the DATETIME format in SAS.

Temporary Data Files

Data files can be either permanent or temporary. A permanent file exists across multiple sessions, whereas a temporary file is available for the current session only. Within a session, the software handles temporary data files and permanent data files identically; you can create graphs and perform analyses using either type of file. At the end of the session, however, the temporary data files are deleted automatically.

Temporary data files serve many purposes. You can perform data exploration and transformation without overwriting your original data by saving the permanent file as

a temporary file at the beginning of the session. Using the temporary file for subsequent analyses prevents accidental corruption of the (permanent) raw data. Furthermore, temporary files simplify file management. Using these files allows the software to eliminate intermediate results, preventing your folders from becoming a depository for unneeded files. For example, you can compare nonparametric smoothers by appending temporary files containing the estimates for each smoother together in a single permanent file from which you can create a scatterplot overlaying the individual smoothers. Having the permanent file of all the estimates eliminates the need to maintain the individual smoother estimates in separate files. By making the files temporary, you permit the software to eliminate these files instead of having to manually delete them yourself.

Creating Temporary Data Files

Temporary data files can only be created using the command language. To generate a temporary data file, use the DWORK command:

```
DWORK filename
```

DWORK can be used in place of any DSAVE command to create a temporary file. For example:

```
DSAVE FILE1
```

As an alternative, you can use DWORK:

```
DWORK FILE2
```

Similarly, in statistical modules, WORK and SAVE commands save the data files. For example, many statistical modules include a SAVE command with an option to specify the statistics saved in the file, such as:

```
SAVE FILE1/RESID
```

As an alternative, you can use WORK:

```
WORK FILE2/RESID
```

Both commands create a data file containing the residuals. However, *file2* will be deleted when you exit the software.

The software saves temporary files in a working folder defined by the File Locations tab of the Options dialog box. To prevent overwriting of a permanent data file with a

8.5.2010

temporary file, we recommend that the working folder differ from the folders assigned for opening and saving data files.

Accessing Temporary Data Files

You can open temporary files using the Open dialog box or the USE command. When using the latter method, you can specify the full path and the name of the file:

```
USE C:\temp\myfile
```

or just the file name:

```
USE MYFILE
```

If you omit the path from the filename, the software searches for the file beginning in the folder for input data files. If the file cannot be found in that folder, the search progresses to the folder for temporary data files. If the file still cannot be found, the search concludes in the folder for output data files. Because files in different folders may share a common name, omitting the path may result in opening a file from the wrong folder. For example, you may have the permanent data file MYFILE in the Data folder and a temporary data file named MYFILE in the C:\temp folder. If the Data folder corresponds to the folder assigned to input data files, submitting:

```
USE MYFILE
```

opens the permanent file because it is the first file having that name found in the search.

To immediately access a specific file in the working folder, precede the filename with the reserved token &WORK.

```
USE &WORK\filename
```

During command processing, the software automatically replaces &WORK with the path corresponding to the folder for working files, eliminating the need to search through multiple folders. In the example above, assuming the folder for temporary files corresponds to C:\temp, submit:

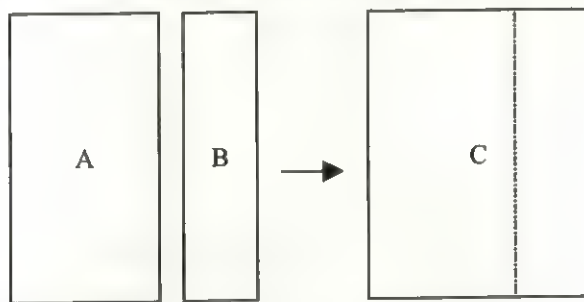
```
USE &WORK\myfile
```

to open the temporary file. Of course, you can always enter the full path to the file instead of using the &WORK token.

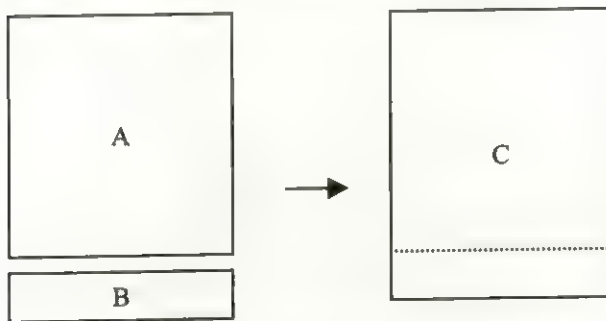
Merging and Appending Data Files

There are two ways to concatenate files: *horizontally* (side by side, joining different variables for the same cases) i.e., to append variables and *vertically* (end to end, adding more cases for the same variables) i.e., to append cases.

Append Variables performs horizontal concatenation:



Append Cases performs vertical concatenation:



You can merge or append only two files at one time. If you have more than two files, merge them successively, two at a time, until they are all part of one file.

Append Variables

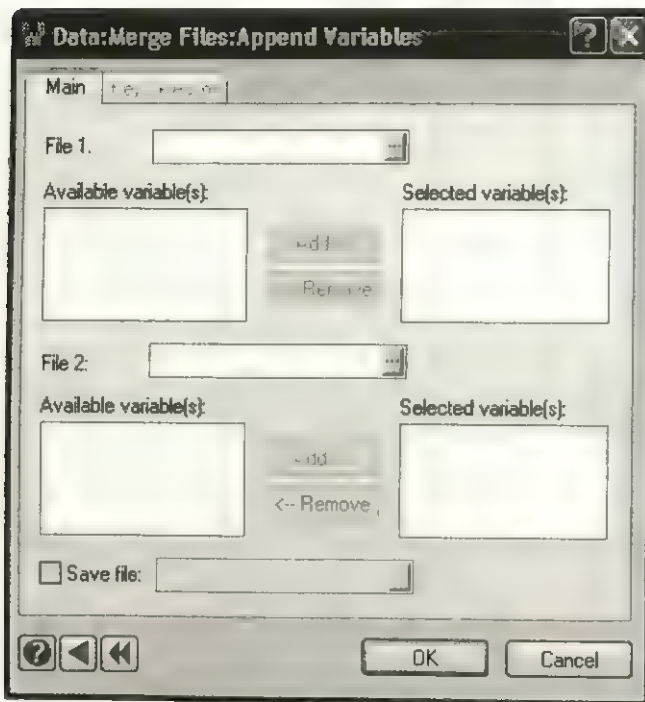
Append Variables joins two SYSTAT files horizontally (side by side). SYSTAT joins the cases from the two files in order, matching the first case from each file, the second case from each file, and so on.

To open the Append Variables dialog box, from the menus choose:

Data

Merge files

Append variables...



For each file, click the File browse buttons to specify each of the files to be merged. If one file has more observations than the other, SYSTAT assigns missing values to the variables from the shorter file for all of the unmatched observations. If the same variable name appears in both files, SYSTAT names the variables trailed by the name of the files with an underscore, for example *var1_file1* and *var1_file2*. Also, if the filenames are same and are at different locations then along with the filename it is also trailed by 1 and 2 to differentiate the two variables, for example *var1_file1_1* and *var1_file1_2*.

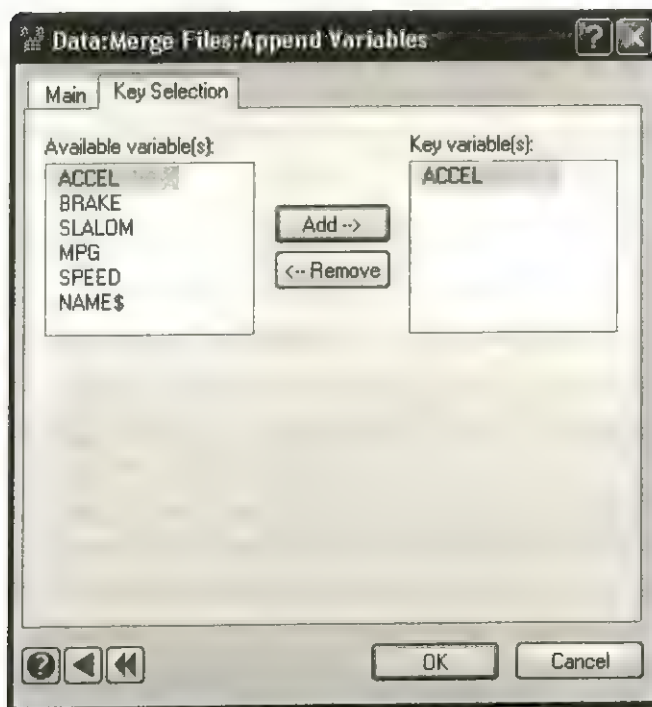
Optionally, you can select the variables from each file to be included in the merged data file. If no variables are selected, all variables from both files are included.

Save file. Specifies an output file for the merged data. If you do not select this option, SYSTAT saves the data in a temporary file.

You can specify one or more key (index) variables by clicking the Key Selection tab.

Merging by Key Variables

You can merge files using one or more key (index) variables. In a drug study, for example, you might place demographic data for the patients in one file and laboratory test results in another file. If both files contain the patients' ID numbers, you could use the ID as the key variable to link each patient's demographics with his or her test results.



Click a variable name to select the variable and click the Add button to use the variable as a key. SYSTAT matches the cases that have the same values for the key variable(s) and joins them in a case in the new file. If there are values for the key variable(s) in one

file and not in the other, the merged file records missing values for the variable whose file did not have values.

In the following example, we start with files *A* and *B* and create file *C*:

File A		File B		File C		
Key	X	Key	Y	Key	X	Y
1	10	1	100	1	10	100
2	20	3	300	2	20	.
				3	.	300

Generating Replicates of a Record

One key variable can have many occurrences of a value that appears only once in the other file. For example, you may need to join information about a mother to the data records for each of her children. For this, SYSTAT replicates the values from the *MOMS* file. For example, using the *FAMILY ID* as the key,

```
MERGE KIDS MOMS / FAMILY
DSAVE PAIRS
```

yields the file represented in the final three columns in the following table:

File KIDS		File MOMS		File PAIRS		
FAMILY	X	FAMILY	Y	FAMILY	X	Y
1	10	1	100	1	10	100
1	1	2	140	1	1	100
1	12			1	12	100
2	6			2	6	140
2	9			2	9	140

Append Cases

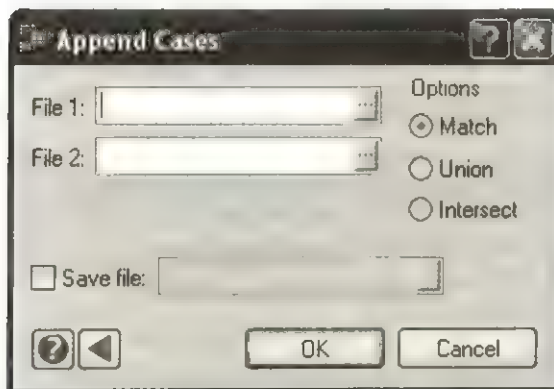
Append Cases joins two SYSTAT files vertically (end to end). SYSTAT places cases from the second file you name below those from the first.

To open the Append Cases dialog box, from the menus choose:

Data

Merge files

Append cases...



Click the File browse buttons to select the files to be appended. One of the following three options must be specified:

- **Match.** Both files must contain the same variables in the same order. If not, SYSTAT displays an error message.
- **Union.** Includes all variables from both input files.
- **Intersect.** Includes only those variables that appear in both files.

For example, if *file1* has variables *a*, *b*, and *d* and *file2* has variables *a*, *b*, and *c*:

- **Match** displays an error message because the two files do not contain the same variables in the same order.
- **Union** writes a new file with variables *a*, *b*, *c*, and *d*. Values of *c* are missing value codes for *file1*, and values of *d* are missing value codes for *file2*.
- **Intersect** includes only variables *a* and *b* in the new file.

Save file. Specifies an output file for the appended data. If you do not select this option, SYSTAT saves the data in a temporary file.

Using Commands

Opening files. You can open an existing SYSTAT data file with the USE command:

```
USE filename
```

For all other types of files, including spreadsheet, database, and ASCII text files, use the IMPORT command:

```
IMPORT filename / TYPE=filetype
```

where *filename* is the name of the input data file that was written by one of the specified applications, and *filetype* is the type of file you are importing (ASCII, SPSS, SAS, SASTRANSPORT, Lotus, Lotus2, Excel, SigmaPlot, dBase, DIF, S-PLUS, MINITAB, STATISTICA, STATA, JMP, StatView or ArcView).

For ODBC import, use the IMPORT command without a filename:

```
IMPORT / TYPE=ODBC CONNECT='connect string'  
      TABLE='table name' VARIABLES='varlist'  
      SQL='statement'
```

The CONNECT, VARIABLES, and SQL options identify the database and table to access, the variables to import, and optional SQL clauses to apply during import.

It is recommended that you specify a file to save after IMPORT:

```
IMPORT test / TYPE=ASCII  
DSAVE NEWTEST
```

Saving files. You can save a SYSTAT data file with the DSAVE and DWORK commands:

```
DSAVE filename / TYPE=filetype
```

and

```
DWORK filename / TYPE=filetype
```

DSAVE creates a permanent data file. DWORK creates a temporary data file that is deleted at the end of the session. For *filetype*, specify RECTANGULAR (default), SSCP, COVARIANCE, CORRELATION, DISSIMILARITY, or SIMILARITY.

To save data in a different format, such as Excel or dBASE, use EXPORT:

```
EXPORT filename / TYPE=filetype
```


where *filename* is the name of the exported data file you are creating, and *filetype* is the type of file you are exporting (ASCII, SAS, SASTRANSPORT, Lotus, Lotus2, Excel, dBase, DIF, SPSS, PLS (for S-PLUS), MTW (for MINITAB), STA (for STATISTICA), DTA (for STATA), JMP, or SVD (for StatView)). SYSTAT automatically assigns the appropriate extension to the filename depending on the specified file type.

Concatenating files. To append variables of files, the command syntax is:

```
MERGE filename1 (varlist1) filename2 (varlist2) / keyvarlist
```

where *varlist1* and *varlist2* are optional—use them to select a subset of the variables in any order you want. The list of key variables is also optional.

To append cases of files:

```
APPEND filename1 filename2 / MATCH or UNION or INTERSECTION
```

To save the appended files permanently, use DSAVE after APPEND.

Spaces in filenames or paths. Filenames or paths to files that include a space must be enclosed in quotation marks for all commands that involve manipulating files. For example, to open *myfile* from the \Program Files\Systat\ folder on the C: drive, specify:

```
USE "C:\Program Files\Systat\myfile"
```

Examples

Example 1

Selecting a Subset of the Variables

You can select a subset and reorder variables when you merge:

```
MERGE MOE(X Y Z) JOE(C A B)
MERGE MOE(Z) JOE
MERGE MOE JOE(A B)
```

The first example merges two files by appending variables, extracting variables *x*, *y*, and *z* from *MOE* and variables *c*, *a*, and *b* from *JOE*. The second selects variable *z* from *MOE* and all of the variables in *JOE*. The third selects all of the variables from *MOE* and *a* and *b* from *JOE*.

Example 2

Side-By-Side Merge

This example demonstrates merging two files by appending variables. One file, *NAME*, contains the names of men who have been presidential candidates in the variable *NAME\$*. The second file, *PARTY*, contains their party affiliations in the variable *PARTY\$*:

<i>NAME</i> data file	<i>PARTY</i> data file
NAMES	PARTYS
Eisenhower	Republican
Stevenson	Democrat
Kennedy	Democrat
Goldwater	Republican
Johnson	Democrat
Humphrey	Democrat
McGovern	Democrat
Nixon	Republican
Ford	Republican
Carter	Democrat
Reagan	Republican
Bush	Republican
Clinton	Democrat

A one-to-one correspondence exists between the cases in the two files. The first case from the *NAME* file corresponds with the first case in the *PARTY* file. To merge these files:

```

MERGE NAME PARTY
DSAVE CANDIDAT
LIST

```

Since no variables are specified, all variables from both files are included in the merged file.

Case	NAME\$	PARTY\$
1	Eisenhower	Republican
2	Stevenson	Democrat
3	Kennedy	Democrat

4	:	Goldwater	Republican
5	:	Johnson	Democrat
6	:	Humphrey	Democrat
7	:	McGovern	Democrat
8	:	Nixon	Republican
9	:	Ford	Republican
10	:	Carter	Democrat
11	:	Reagan	Republican
12	:	Bush	Republican
13	:	Clinton	Democrat

Example 3 ***Merging with a Key Variable***

This example merges the files *CANDIDAT* and *ELECTION* by the variable *NAME\$*. *CANDIDAT* was created by merging *NAME* and *PARTY* files. The data in the *ELECTION* file are shown below.

NAMES	LOSERS	YEAR
Eisenhower	Stevenson	1952
Eisenhower	Stevenson	1956
Kennedy	Nixon	1960
Johnson	Goldwater	1964
Nixon	Humphrey	1968
Nixon	McGovern	1972
Carter	Ford	1976
Reagan	Carter	1980
Reagan	Mondale	1984
Bush	Dukakis	1988
Clinton	Bush	1992

To merge the two files, specify *NAME\$* as the key variable.

```

MERGE ELECTION CANDIDAT/NAME$
DSAVE MERGFILE
LIST

```


The resulting file includes:

Case	NAME\$	LOSER\$	YEAR	PARTY\$
1	Bush	Dukakis	1988.000	Republican
2	Carter	Ford	1976.000	Democrat
3	Clinton	Bush	1992.000	Democrat
4	Eisenhower	Stevenson	1952.000	Republican
5	Eisenhower	Stevenson	1956.000	Republican
6	Ford	.	.	Republican
7	Goldwater	.	.	Republican
8	Humphrey	.	.	Democrat
9	Johnson	Goldwater	1964.000	Democrat
10	Kennedy	Nixon	1960.000	Democrat
11	McGovern	.	.	Democrat
12	Nixon	Humphrey	1968.000	Republican
13	Nixon	McGovern	1972.000	Republican
14	Reagan	Carter	1980.000	Republican
15	Reagan	Mondale	1984.000	Republican
16	Stevenson	.	.	Democrat

The missing values occur where entries in the candidate file have no matching value for NAME\$ in the election data file (for example, Ford, Goldwater, Humphrey, etc., did not win). The election file has three names that have more than one entry: Eisenhower, Nixon, and Reagan. In these cases, SYSTAT replicates the corresponding values from the candidate file.

Note: If you select a subset of variables while merging, you need not include the key variable(s) in both subsets (of course, they must be in both files). The following, for example, is valid:

```
MERGE INDOOR (TIME, LOC, CO2) OUTDOOR (NOX) / TIME, LOC
```

Example 4 End-to-End Append

The following are two SYSTAT files, named *MEN* and *WOMEN*:

<i>MEN</i> data file		<i>WOMEN</i> data file	
SEX\$	AGE	SEX\$	AGE
Male	18	Female	23
Male	35	Female	40
Male	24	Female	40
Male	20	Female	31

To append them end to end:

```
APPEND WOMEN MEN
DSAVE SEXES
```

SYSTAT places the cases from *WOMEN* before those from *MEN* because *WOMEN* are listed first in the APPEND command.

Case	SEX\$	AGE
1	Female	23.000
2	Female	40.000
3	Female	40.000
4	Female	31.000
5	Male	18.000
6	Male	35.000
7	Male	24.000
8	Male	20.000

References

- Bowman, J.S., Emerson, S.L., and Darnovsky, M. (1996) *The practical Sql handbook : Using structured query language*. New York: Addison-Wesley.
- Date, C.J. and Darwen, H. (1997). *A guide to the Sql standard : A user's guide to the standard database language Sql*. New York: Addison-Wesley.
- Melton, J. and Simon, A.R. (1993). *Understanding the new Sql : A complete guide (The Morgan Kaufmann Series in data management systems)*. San Francisco: Morgan Kaufmann.

Entering and Editing Data

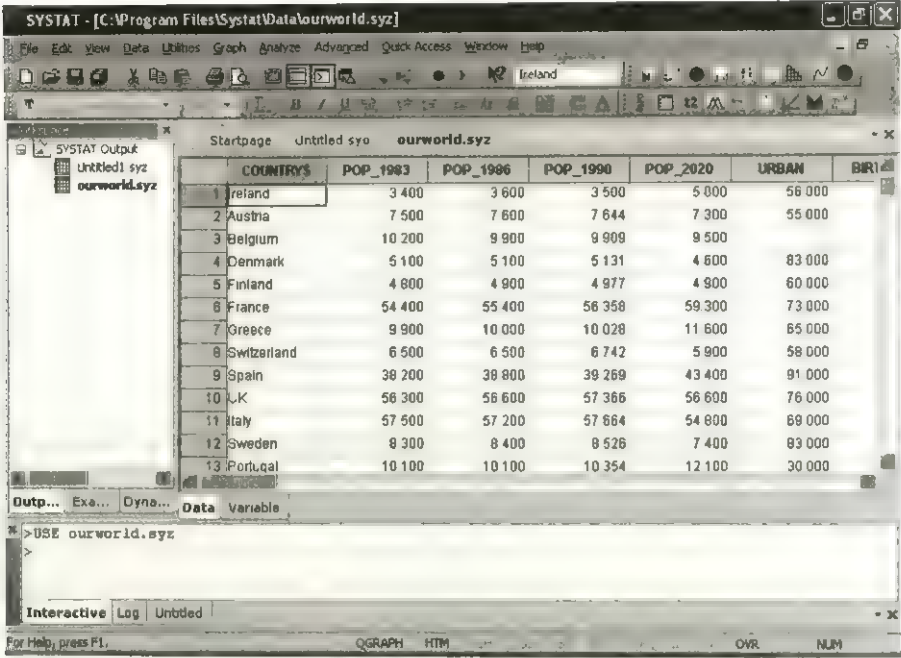
Leland Wilkinson, Laszlo Engelman, and Mark Bjerknes

The Data Editor is used for entering, editing, and saving data. Entering data is a straightforward process—simply type the data. SYSTAT assigns default variable names (which can be changed if desired) to each column.

Editing data includes changing variable names or attributes, adding and deleting cases or variables, moving variables or cases, and correcting data errors. In addition, you can locate specific variables, find cases that meet specific conditions, or identify each case in the Data Editor using a variable's values.

Data Editor

Viewing, entering, and editing data occurs in the Data Editor. You can also run transformations and view the results, select subsets of cases, and transpose, append, and merge data files as described in this and the following chapters.



To display the Data Editor, choose Data Editor from the View menu or click the Data Editor tab in the Viewspace.

Data File Structure

SYSTAT uses data organized in rows and columns. The rows are cases and the columns are variables. A case contains information for one unit of analysis, such as a person, an animal, a business, or a jet engine. Variables are the information collected for each case, such as age, body weight, profits, or fuel consumption.

For example, a portion of the data from the file *OURWORLD* is shown. Each row (case) has data for one of 15 countries, and the columns (variables) include the name of the country, population in 1990, gross domestic product per capita, years of life expectancy estimates for females, literacy, type of country, and the number of McDonald's restaurants.

<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>5</i>	<i>6</i>	<i>7</i>
Austria	7.644	13500.299	80	98	Europe	30.0
Belgium	9.909	13724.502	80	98	Europe	12.0
Denmark	5.131	14363.064	79	99	Europe	16.0
Finland	4.977	14947.788	80	100	Europe	9.0
France	56.358	14542.657	82	99	Europe	193.0
Greece	10.028	5614.184	80	95	Europe	1.0
Ireland	3.500	8970.885	78	99	Europe	14.0
Switzerland	6.742	17723.499	83	99	Europe	25.0
Spain	39.269	10153.121	82	97	Europe	42.0
UK	57.366	14259.401	79	99	Europe	400.0
Italy	57.664	13930.604	81	93	Europe	14.0
Sweden	8.526	15563.332	81	99	Europe	56.0
Portugal	10.354	6963.158	78	83	Europe	2.0
Netherlands	14.936	13785.455	81	99	Europe	69.0
WGermany	62.168	15211.956	81	99	Europe	391.0

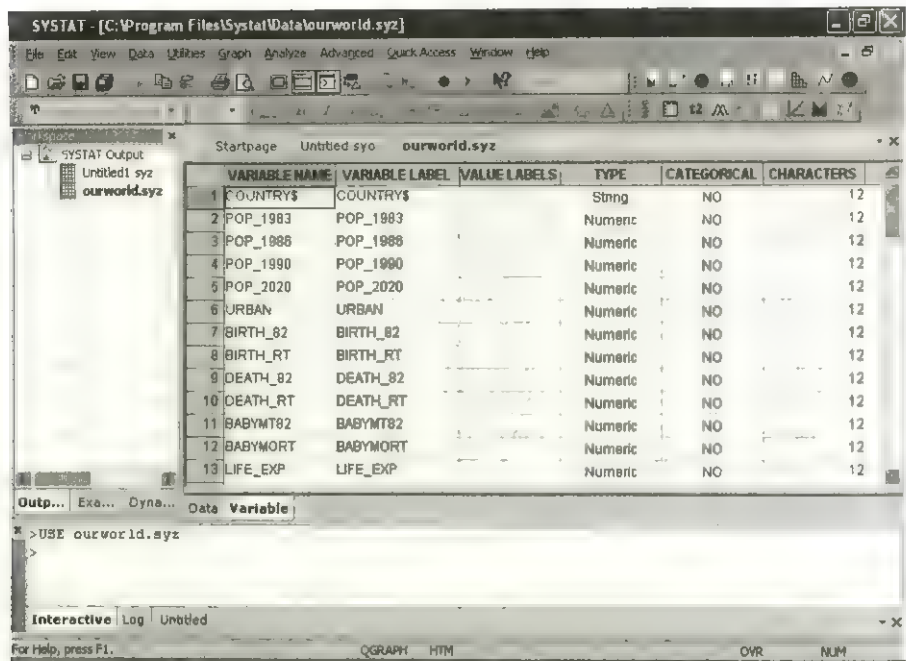
When data are arranged in rows and columns (as in these data), and then stored in a file, we call the file a cases-by-variables or rectangular data file.

Variable Editor

The Data Editor allows you to edit data values directly in the grid. The Variable Editor allows you to edit the properties of variables. It has one row corresponding to each variable, and the row includes all the items that are in the Variable Properties dialog. With it, you can set any of the properties for any variable with a single click of the mouse. The Variable Editor also has a section where you can view and set the processing conditions in effect for the current data set. For details on processing conditions, refer to the section “Processing Conditions in Effect” on page 41.

To view the Variable Editor, from the menus, choose:

View
Variable Editor...



Variable name. You can assign a name to every variable. Variable names are used to specify analyses and to label output. Variable names can contain up to 256 letters or numbers and must begin with a letter. Names of character or string variables must end with a dollar sign (\$), which counts as a character. SYSTAT defines all variables whose names do not end in \$ as numeric. Use the underscore character (_) to indicate a space within a name. The only other characters that can be used in a variable name are parentheses, which denote subscripted variables.

Variable names, unlike character values, are not case sensitive. If you type govern, SYSTAT automatically uses GOVERN.

Variable label. A Variable label is a string or text associated with the variable. It is an alias for the variable name. It can be up to 256 characters in length, and in contrast to the variable name, can contain spaces and special characters, and can start with any character. By default, the label for a variable is its name itself. You can define a variable label using either the Variable Editor, the Variable Properties dialog box, or the VARLAB command. If an analysis is performed using a given variable, its variable label is displayed instead of its name in the Output Editor. You can control the display of variable labels in the output using the Edit:Options dialog or the VDISPLAY command.

To view the variable labels as a tool tip, pause the mouse pointer on it. Any defined variable labels will be saved in the data file.

Value label. Value labels are the labels given to the values of a variable. You can define value labels using either the Variable Editor, the Value Labels dialog, or the LABEL command. To view value labels in the Data Editor, right-click on a variable name and select View Value Labels. By default, value labels will appear in the output including in graphs. You can control the display of value labels in the output using the Edit: Options dialog or the LDISPLAY command. To view the value label as a tool tip, pause the mouse pointer on it. Any defined value labels will be saved in the data file.

Variable type. Choose the variable type. SYSTAT accepts numbers and characters as data.

- **Numeric.** Use for variables that contain numbers only. Numeric variables can contain up to 23 digits. Use a minus sign for negative numbers. You can also use exponential notation for very large or very small numbers. For example, 1.5E4 and 1.5E-4 appear as:

$$\begin{aligned} 1.5E4 &= 1.5 \times 10^4 &= 15000 \\ 1.5E-4 &= 1.5 \times 10^{(-4)} &= .00015 \end{aligned}$$

- **String.** String or character variables contain text information. A string variable can contain up to 256 characters and can contain any common typewriter characters. Specify the number of characters the variable will hold. When storing values of string variables, SYSTAT differentiates between upper and lower case—"europe" is not the same as "EUROPE" or "Europe." If you include numbers within string values, SYSTAT treats them as text; you cannot perform numerical analyses on them. Some valid values are:

male	New York	\$)##(\$&%^/#@
female	Chicago	!@#\$%^&*)?(\$

Categorical. Use to declare a variable as categorical. SYSTAT treats the values of such variables, whether they are numeric or string, as discrete categories. Categorical variable information will be saved in the data file by default. You can prevent this by unchecking Save category variable information to data file in the Data tab of the Edit:Options dialog box.

Characters. Specify the maximum field width of the variable, i.e., the total number of characters to be displayed in the Data Editor. For string variables, you can specify up to 256 characters. For numeric variables you can specify up to 23 characters.

Display. For numeric variables, you can choose among several display options:

- **Normal.** Displays variable values in standard decimal notation—for example, 123.45. Specify the number of decimal places to display.
- **Exponential notation.** Displays variable values in exponential notation. Specify the number of decimal places to display.
- **Date and time.** Select the desired date and time format from the list. The number of letters indicates the number of characters displayed. Two *m*'s (*mm*) represent months as two digits; for example, September would appear as 09. Three *m*'s (*mmm*) represent month abbreviations, so September would be "Sep". To display the entire month name, use four *m*'s (*mmm*). Use the same rules for days of the week, using *d*'s instead of *m*'s. For instance, *ddd* displays "Tue". Use 'yy' to display the last two digits of the year, 'yyyy' to display all the digits of the year, 'hh' to display time in hours, 'mm' in minutes, and 'ss' in seconds. The 'hh' display format uses 12 hour scale. If you want to display the 24 hour scale, use the upper case, i.e., 'HH'. You can select any date and format from the dropdown list. You can also use any custom format by simply double-clicking and typing the desired format. For example, you can specify the following formats:

```
dd:MMM:yyyy-HH:mm:ss
MMM dd,yyyy
```

Display options are specific to *variables*, not cells. For any given variable, all cells have the same format.

Decimals. Specify the number of decimal places to be displayed in the Data Editor. This option is applicable for numeric variables when the display option is either Normal or Exponential notation. You can specify up to 14 digits. The number of decimal places should be less than the corresponding field width. The default is the number specified in the Output tab of the Edit:Options dialog box.

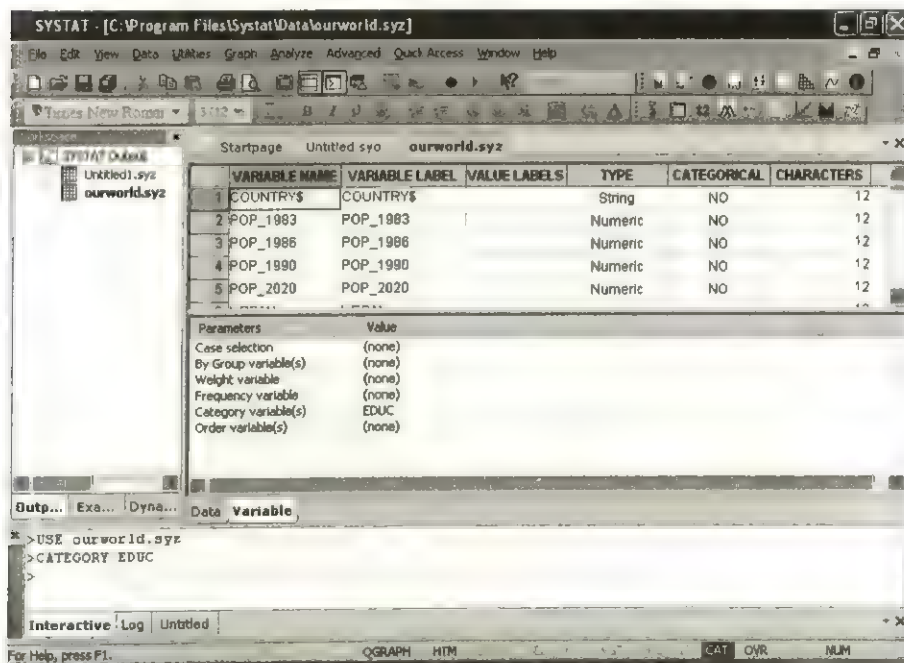
Pasting Variable Properties

The Variable Editor not only allows you to edit the variable properties manually, but also allows you to edit them through the right-click options like copy and paste. So, you can copy properties of one or more variables and paste them for the desired variables. You can copy properties like variable label, value labels, characters, decimals, display format, date and time format, and variable comments. You can paste a property of one variable to the same property of another variable.

Note that you cannot paste the variable type and category from one variable to another. You can also rearrange the variables and create new ones through the context menu of rows in the Variable Editor. The functioning of these options is similar to that of the context menu options for variable names in the Data Editor. For more information on these options, refer to the section “Right Click Editing” on page 56.

Processing Conditions in Effect

The Variable Editor in SYSTAT contains a Processing Conditions section, which displays the processing conditions in effect for the current data set. The processing conditions are weight, frequency, category, order and grouping variable specifications, and case selection conditions. You can change a processing condition by double clicking on it, and making the necessary changes in the dialog that opens.

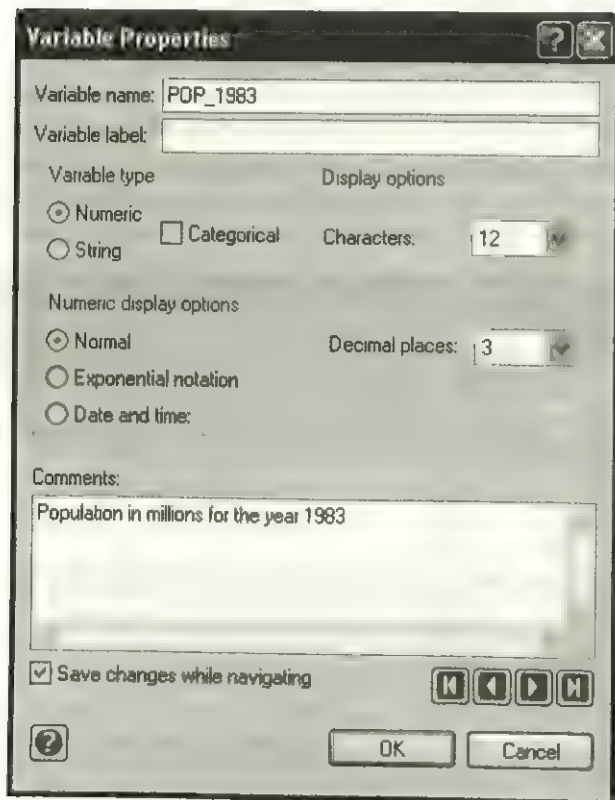


Variable Properties Dialog Box

To set the properties of individual variables, use the Variable Properties dialog box. To open the Variable Properties dialog box, from the menus choose:

Data

Variable Properties...



Notice that you can open the Variable Properties dialog box from the menus only when the Data Editor tab is active.

You can navigate from one variable to the next while setting the properties. To navigate, click on the navigation buttons at the bottom of the dialog box. From a given variable, you can go to the next, previous, first or last variable in the data file with a single click of the mouse. By default, changes to properties of any variable are saved while navigating. If you do not want to save the changes unless you press OK, you can uncheck the Save changes while navigating option.

To Specify Variable Properties

In the Variable Editor

- From the menus choose:

View
Variable Editor...

or

- Click on the Variable tab of the active data file.

or

- Double-click on a variable name in the Data Editor.

or

- Right-click on the Data tab of the active data file and select Data/Variable Editor.

In the Variable Properties Dialog Box

- From the menus choose:

Data
Variable Properties...

or

- Right-click on a variable name in the Data Editor and select Variable Properties.

To display the list of variable(s) in the file along with properties, choose from the menus:

Utilities
File Information
Dictionary...

To display only the variable names, select Names.

Subscripted Variables

An easy way to specify a subset of variables is to use subscripted variables. For example, if you have 10 questions, you could simply name them $Q(1)$, $Q(2)$, ..., $Q(10)$ or, for string variables, Q(1)$, Q(2)$, ..., Q(10)$. You can then refer to the range of variables using subscript notation, such as:

```
CSTATISTICS Q(1 .. 10)
```

to get descriptive statistics for the responses to all 10 questions. SYSTAT will perform analyses on all of the variables from $Q(1)$ to $Q(10)$, regardless of the order in which they appear in the file.

Subscripted names can be used for numeric or string variables. The total length of the variable name can be up to 256 characters, including the parentheses (and the dollar sign for character variables).

Grouping Variables

A grouping variable contains a value that identifies group membership for each case. The values of these variables are used to separate the cases into groups (or cells) in subsequent analyses and graphs. In the *OURWORLD* data file, *GROUP\$* is a string grouping variable, and *GROUP* is a numeric grouping variable that gives each variable the following identification: 1 for *Europe*, 2 for *Islamic*, and 3 for *New World*. A numeric grouping variable named *SEX* might contain the number 1 for males and 2 for females, while a string grouping variable such as *SEX\$* could have the values *Male* and *Female*.

Variable Comments

The Variable Editor contains a column to record information about each variable. There is no limit to the amount of information you can record in a variable's comments. You can also record comments using the Variable Properties dialog box. Issue the *NAMES / COMMENT* command to send the specified variable comments to the output.

The comments of a variable can be viewed as a tool tip in the Data/Variable Editor. To view, pause the mouse pointer on the variable name.

Missing Data

Some cases may be missing data for a particular variable—for example, a subject might not have a middle name, or a state might have failed to report its total sales. In the Data Editor, missing numeric values are indicated by a period, and missing string values are represented by an empty cell.

Arithmetic that involves missing values propagates missing values. If you add, subtract, multiply, or divide when data are missing, the result is missing. If you sort your cases using a variable with missing values, the cases with values missing on the sort variable are listed first. If you specify conditions and a value is missing, SYSTAT sets the result to missing. For example, if you specify:

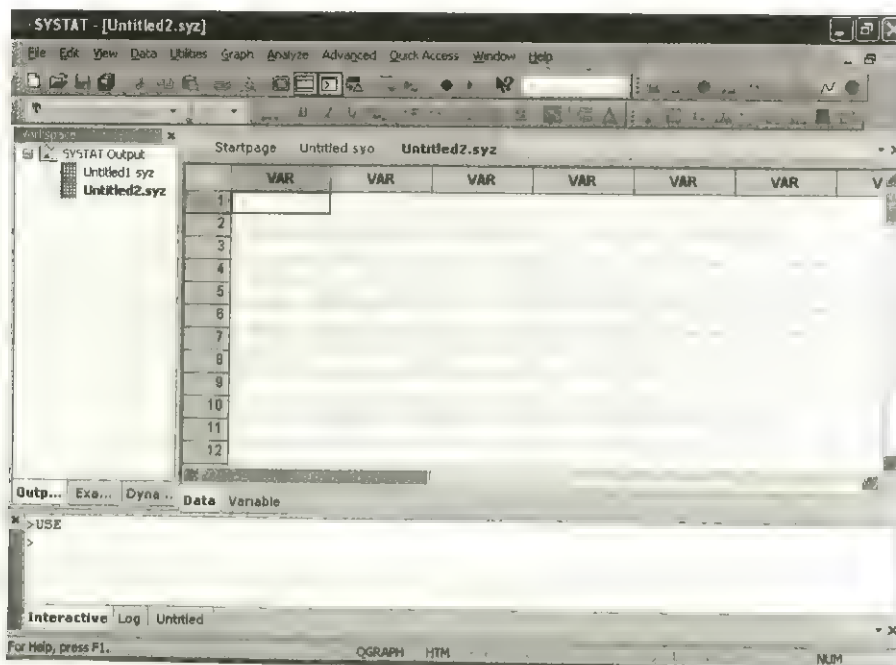
```
IF age > 21 THEN LET age$='Adult'
```

and *AGE* is missing, the value of *AGE\$* is set to missing. To perform an analysis on only those cases with no values missing, use **SELECT COMPLETE** prior to the analysis.

Note: If you are entering data in an ASCII text file, enter a period (.) to flag the position where a numeric value is missing. Where character data are missing in an ASCII text file, enter a blank space surrounded by single or double quotation marks.

Data Entry

Entering data in the Data Editor is a straightforward process. Simply type the data, and after typing each value, press Enter.



SYSTAT denotes empty columns with the name VAR. Upon the entry of a value in a column, SYSTAT assigns the name VAR_ *n* to that variable, where *n* corresponds to the variable number. SYSTAT fills all empty columns to the left of this variable with missing values and assigns default variable names. If the entered value contains characters, SYSTAT appends a \$ to the variable name.

The active (current) cell is highlighted with a thin blue border. You can enter data by case (one row at a time) or by variable (one column at a time). To specify whether the active cell moves across (for case entry) or down (for variable entry) when you press Enter, choose Options from the Edit menu, click the Data tab, and select the desired behavior.

Data values are not recorded until you press Enter or select another cell. To clear the Data Editor, making it ready for new data, select

```
File
  New
    Data...
```


Note that there is no limit to the number of variables or cases your data file can contain. However, the amount of memory available on your system may impose a practical limit.

Editing Data

You can edit data to correct errors, change variable names, or add, delete, or move variables or cases. SYSTAT provides a number of tools to help you edit your data. You can locate specific variables, find cases that meet specific conditions, or specify an ID variable to identify each case in the Data Editor.

Tip: Any changes you make to the data file are not permanent unless you save the data file.

Data Editor Navigation

You can use either the mouse or the keyboard to move around in the Data Editor and to select cells, cases, or variables.

- Click the mouse in any cell to select the cell.
- Use the scroll bars on the bottom and right side of the editor to scroll through the data to see more cases and variables. Click the scroll arrows to move one case or variable at a time, or drag the icon to move to a new location in the data.
- Using the keyboard, you can move in all directions using the arrow keys. You can also move the active cell using the Enter and Tab keys alone or in combination with the Shift key.

Press:	To move the active cell:
Enter	Right one cell or down one cell. Specify the behavior using the Data tab of the Edit:Options dialog box.
Shift+Enter	Left one cell or up one cell. Moves the active cell in the reverse direction of the Enter key.
Tab or right arrow	Right one cell.
Shift+Tab or left arrow	Left one cell.
Down arrow	Down one cell.
Up arrow	Up one cell.
Home	Beginning of the current row.
End	End of the current row.
Page Up	Up one page.
Page Down	Down one page.
Ctrl+up arrow	Top of current column.
Ctrl+down arrow	Bottom of current column.
Ctrl+left arrow	First cell of current row.
Ctrl+right arrow	Last cell of current row.
Ctrl+Home	First cell (top left).
Ctrl+End	Last cell (bottom right).

Selecting Cells

You can select cells using either the mouse or the keyboard.

Using the Mouse

- Drag the mouse to choose a range of adjacent cells, variables, or cases.
- Click on the case number at the left side of a row or the variable name at the top of a column to select the entire row or column.
- Hold down the Shift key as you click on any cell, case number, or variable name. The resulting selection depends on the initial state of the Data Editor and the location of the mouse click.

Initial state	Shift-Click Target	Resulting selection
Selected cell (active cell in row I and column J)	Cell	Range of cells between row I and target cell row, and between column J and target cell column.
	Case number	Target row.
	Variable name	Target column.
Selected row (I)	Cell	Range of cells between row I and target cell row, and between first column and target cell column.
	Case number	Rows from row I to the target row.
	Variable name	Columns from column 1 to the target column.
Selected Column (J)	Cell	Range of cells between first row and target cell row, and between column I and target cell column.
	Case number	Rows from row 1 to the target row.
	Variable name	Columns from column J to the target column.

Using the Keyboard

- Hold down the Shift key and use the arrow keys to extend the selection from the current cell to the target cell defined by that arrow key. For example, Shift and the left-arrow key selects the active cell and the cell immediately to the left.
- You can also use Shift with some shortcut key combinations; for example, Ctrl+Shift+End extends the selection to the bottom right corner.

Correcting Data Values

You can easily replace data values in the Data Editor. Simply select the cell, edit the value, and move to another cell using the mouse or keyboard.

Editing a cell often involves changing portions of an entry instead of replacing the entire value. You can edit in a cell.

In-place editing. Double-click the cell whose value you wish to edit. Double-clicking within the cell value places the insertion point at begin. This insertion point can be moved using the following keys:

Key	Moves the insertion point:
Left arrow	One character left.
Right arrow	One character right.
Home	Beginning of value.
End	End of value.

If you select a portion of the value by dragging the mouse over it, SYSTAT deletes the selected portion before inserting the typed values. Note that, in the display format, SYSTAT may not show the number completely but when we double-click on a cell, the number is completely shown in the cell. For example, suppose a cell has an entry of 123.3654329. If the display format for that variable contains three decimal places only, the number appears as 123.365 in the cell but when we double click SYSTAT shows 123.3654329, the complete value.

Single-click replacement. To replace an entire cell entry, single-click the cell and type the new entry. In contrast to in-place editing, the replacement value is not constrained by the display format. However, shifting the active cell to another location formats the replacement value in accordance with its column. For example, suppose a variable has a display format of eight characters with two decimal places. We want to replace the entry 111.11 with 222.2222, so we single-click the cell and type the new entry. Moving to another cell formats the entered value to 222.22. To edit the final two decimal places, we must use the in-place editing.

Note: No changes to a cell entry become permanent until another cell becomes active. Use the Escape key (Esc) before moving to another cell to cancel changes and revert a cell entry to its original value.

Editing Dates and Times

To change a date or time, click the cell. Up and down arrows appear at the right side of the cell. Use these arrows to increase or decrease a selected portion of the value by one unit.

For example, suppose we want to change the date 06/10/2003 to 08/14/2003. Click the cell, choose the month, and press the up arrow two times. To change the day, select the original date (06/10/2003), and press the up arrow four times. Alternatively, you can change the date by selecting the fields and typing the new value. For instance, choose 06 and type 08.

Changing Variable Names and Types

You can change variable names and types using the Variable Editor. To change a variable name, click on the existing variable name in the Variable Name column and type a new name. Double-click if you just want to edit the existing name.

You can also change variable names using the Variable Properties dialog box (or the DEFVAR command). To do this, right-click on a variable name and select Variable Properties.

Changing variable names. Type the new name. String variables must end in \$.

Changing variable types. You can change a variable from numeric to string (and vice versa).

- **String to numeric.** If the variable contains non-numeric data, SYSTAT converts these entries to missing values. Cells with numeric data retain their numeric information with good accuracy. Cells with a combination of numeric and string data are converted to missing values.
- **Numeric to string.** SYSTAT uses the display format for the character entries.

Adding Variable(s) or Case(s)

You can add new variables and/or cases to a file in the same way you entered the original data.

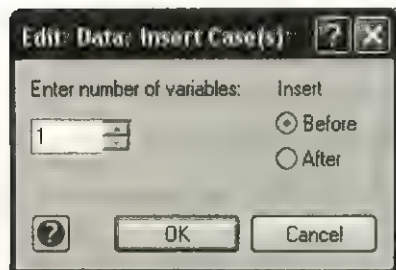
- To add a new variable using the Variable Editor, scroll past the existing variable names until you see empty rows. Click on the first empty variable name cell, and type a name for the new variable. If you are using the Data Editor, scroll past the existing columns to any empty column, not necessarily the first, with the variable name VAR. Right-click on the name, select Variable Properties, type a name for the new variable or use the default name in the dialog box, and press OK. SYSTAT automatically creates variables corresponding to the preceding empty columns as well.
- To add new case(s), move the cursor to the first empty (all blank) row and type the new data.
- To insert case(s) between existing cases, select a row and right click; then select: Insert case(s)...

Or, from the menus choose:

Edit

Data

Insert Case(s)...



- Choose or type the number of cases you want to insert.
- Select Before or After, to insert new case(s) before or after the selected row respectively.
- Click OK.

The inserted cases contain missing values.

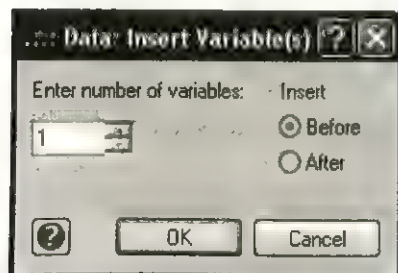
- To insert variable(s) between existing variables, select a column and right click; then select:
Insert variable(s)...

Or, from the menus choose:

Edit

Data

Insert Variable(s)...



- Choose or type the number of variables you want to insert.

- Select **Before** or **After**, to insert new variable(s) before or after the selected variable respectively.
- Click **OK**.

Inserted variable(s) receive default name(s) and are of numeric type. All cells are assigned a missing value.

You can insert cut or copied variables from the clipboard, between existing variables. To insert, right click on a variable name and select **Insert Variable(s) from Clipboard**. For details, refer to the section “Cutting and Pasting Variable(s) or Case(s)” on page 54.

To add variable(s) or case(s) from another file, use **Append Variables** or **Append Cases** from **Merge** files in the **Data** menu.

Dropping Variable(s)

To delete one or more variable(s), select the variables in the **Data Editor** and choose:

Edit
Cut...

Or, from the menus choose:

Edit
Data
Delete Variable(s)...

The remaining variables are then shifted to the left.

The difference between **Cut** and **Delete Variable(s)** is that the former deletes the selected data and also copies the data to the clipboard, so that it can be pasted to a new location, whereas the latter simply deletes the selected data.

Deleting Case(s)

Using the mouse, you can delete case(s) just like you delete variable(s). Select the case(s) in the **Data Editor** and choose:

Edit
Cut...

Or, right-click and select **Cut**.

Or, from the menus choose:

Edit
Data
Delete Case(s)...

SYSTAT removes the specified cases, moving subsequent cases up to fill their places. Be aware that SYSTAT rennumbers the cases after they have been deleted. It is often helpful to create an ID variable that numbers the case(s), to make it easier to track them through various file manipulations.

If you want to delete case(s) based on conditions, use the If Then Let dialog box.

To open the dialog box, from the menus choose

Data
Transform
If..Then Let...

Cutting and Pasting Variable(s) or Case(s)

You can rearrange variable(s) or case(s) by using Cut and Insert Variable(s) from Clipboard from the Edit menu, or the CUT and PASTE commands. For example, to move one or more adjacent variables to a different position in the data file:

- Choose the variable(s) you want to move using the mouse.
- Choose Cut from the Edit menu (or by right-clicking) to remove the variable(s) from their current location and copy them to the clipboard.
- Select the column where you want to paste the variable(s) and select Insert Variable(s) from Clipboard from Data in the Edit menu. The variable(s) are pasted to the left of the selected column.

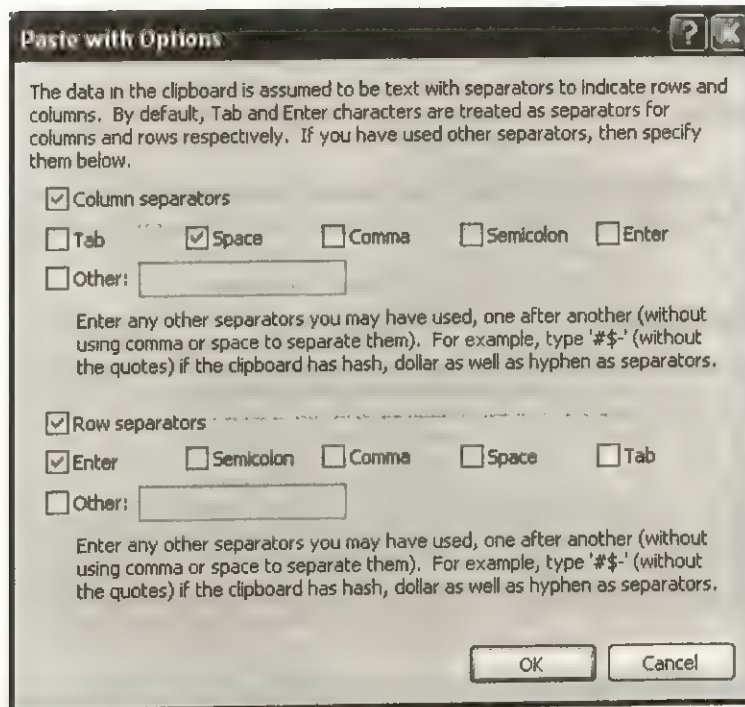
You can also use Copy to copy variable(s) to the clipboard without deleting them from their original location. However, if you try to paste the same variables to another location in the same data file, SYSTAT gives default names to the pasted variable(s), since two variables in the same data file cannot have the same name.

You can use Copy and Insert Variable(s) from Clipboard to copy just about any type of data, and even to copy data from one application to another, within the limits of common sense. In the Data Editor, that means that pasted data must be of the appropriate type and “shape.” You cannot paste a string value into a numeric variable or copy a range of cells and paste them into a single cell.

Paste with Options

Paste with Options is a dialog that will allow you to specify the row and column separators used in a block of ASCII data. Thus, you can now arrange a block of data before pasting it in the Data Editor. To open the dialog, right-click on a Data Editor cell and select Paste with Options. Or, from the menus, choose:

Edit
Data
Paste with Options...



You can choose any of the commonly used separators, namely, the tab, space, comma, semi-colon and the enter character. You can also specify other separators.

Insert Variable(s) from Clipboard

You can insert cut or copied variable(s) from the clipboard in the Data Editor directly, without first having to create new columns. You can insert single or multiple variables within the same data file or in another data file. To insert variables, right-click on a variable name and select **Insert Variable(s) from Clipboard**. Or, from the menus choose:

```

Edit
Data
  Insert Variable(s) from Clipboard...

```

The inserted variables will have the same variable properties as the original variables. However variables inserted after a copy operation will receive default names, since no two variables in the Data Editor can have the same name.

Right Click Editing

When you right-click in the Data Editor you will be presented with several editing options. The behavior of these options depends on the location of the right-click.

Editing Option	Location of right-click in the Data Editor		
	Row Heading	Column Heading	Data Cells
Variable Properties	Menu item does not appear.	Gives the user access to the Variable Properties dialog box. If the variable has already been defined, it brings up the dialog box with the settings for that variable. If multiple variables are selected, the dialog reflects properties of the current variable in the selection.	Menu item does not appear.
Variable Statistics	Menu item does not appear.	Displays the basic statistics and bar graph of the current variable.	Menu item does not appear.
Cut	Cuts the selected case(s) to the clipboard and shifts the succeeding case(s) up by the number of cases cut.	Cuts the selected variable(s) to the clipboard and shifts the succeeding variable(s) to the left by the number of variables cut.	Cuts the content of the selected cells to the clipboard and defines the selected cells as missing values.
Copy	Copies the selected case(s) to the clipboard.	Copies the selected variable(s) to the clipboard.	Copies the content of the selected cells to the clipboard.

Copy as text	Menu item does not appear.	Copies the content of the selected variable(s) to the clipboard, as text (without any separator information). If value labels of a variable are displayed instead of data values, this option copies the value labels of that variable.	Copies the content of the selected cells to the clipboard, as text (without any separator information).
Paste	Pastes the content of the clipboard in or starting from the current window	Menu item does not appear.	Pastes the content of the clipboard over the selected cells.
Paste with Options	Menu item does not appear.	Menu item does not appear.	Opens the Paste with Options dialog where you can specify row and column separators for the data in the clipboard.
Paste Data	Menu item does not appear.	Pastes the content of the clipboard or the data corresponding to copied variable(s) in (starting from) the current column, as data values.	Menu item does not appear.
Paste Properties	Menu item does not appear.	Pastes all the properties of the copied variable(s) as the properties of (starting from) the current variable.	Menu item does not appear.
Insert Variable(s) from Clipboard	Menu item does not appear.	Inserts variables with data from the clipboard to the left of the current variable. This item is disabled when there is no data in the clipboard	Menu item does not appear.
Delete	Sets the selected case(s) to missing values.	Sets the values of the selected variable(s) to missing values.	Sets the content of the selected cells to missing values.
Insert Case(s)	Opens the Insert Case(s) dialog box where you can specify the number of cases to insert. You can insert upto 10 cases at a time. All inserted values are set to missing values.	Menu item does not appear.	Menu item does not appear.
Delete Case(s)	Deletes the selected case(s) from the Data Editor and shifts all succeeding case(s) up by the number of cases deleted.	Menu item does not appear.	Menu item does not appear.

Insert Variable(s)	Menu item does not appear.	Opens the Insert Variable(s) dialog box where you can specify the number of variables to insert. You can insert upto 10 variables at a time. All inserted values are set to missing values.	Menu item does not appear.
Delete Variable(s)	Menu item does not appear.	Deletes the selected variable(s) from the Data Editor and shifts all succeeding variable(s) by the number of variable(s) deleted.	Menu item does not appear.
View Value Labels	Menu item does not appear.	Displays either the already defined value labels or the data values of the current variable. This option is disabled when no value labels are defined for a variable.	Menu item does not appear.

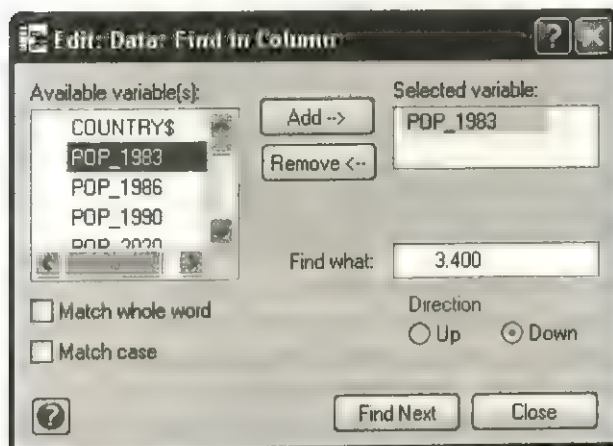
Right-click on the Data Editor tab in the Viewspace

Editing Option	Operation
Copy All	Copies all the contents of the data file to the clipboard.
Set as Active Data File	Sets the current data file as Active, if it is already not. Otherwise, this option is disabled.
Data/Variable Editor	Takes the control to the Variable Editor , if it is in the Data Editor and vice versa.
File Comment	Allows you to give comments and edit it for the current data file. To view the comments given to the current data file, pause the mouse pointer at the top left corner of the Data/Variable Editor.
Print Preview	Shows the Preview of the current data file for printing.
Show Toolbar	Shows /Hides the Toolbar for editing data.
New	Opens a new data file.
Save As	Saves the current data file if it is not active, otherwise this option is disabled.
Options	Opens the Global Options dialog box.
Close	Closes the current data file. If the current data file is active a prompt for saving the data file appears before closing.

Find Values in Column

To search for numbers, strings, or dates in a column, from the menus choose:

Edit
Data
Find in Column...

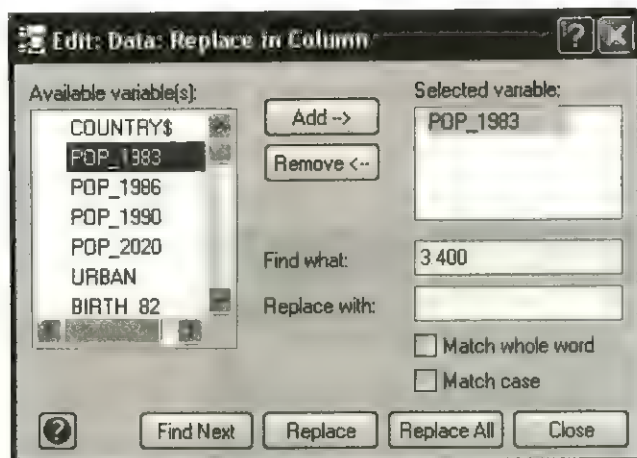


You can search for a string or number as a part or whole of each cell entry in any given column of the data file. You can specify the direction of the search — Up to search from the current location to the first case, — Down to search from the current location to the last case in the column. By default, the search is performed on the current variable; you can change the variable to be searched. To search only for complete entries, check Match whole word. To search for entries with the matching case, check Match case.

Replace Values in Column

To search and replace for numbers, strings, or dates in a column, from the menus choose:

Edit
Data
Replace in Column...

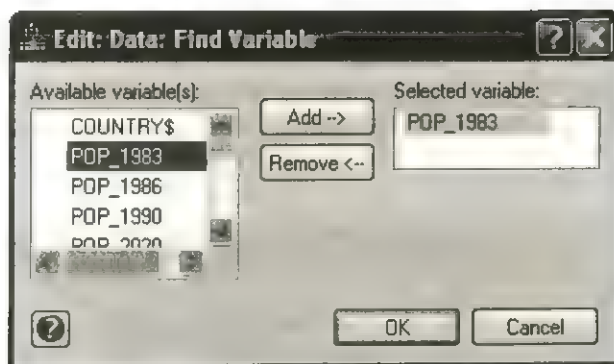


You can search for the specified string or number as a part or whole of each cell entry in a variable, and replace them with the desired string or number. By default, the search is performed on the current variable; you can change the variable to be searched. To search only for complete entries, check **Match whole word**. To search for entries with the matching case check **Match case**. Press **Replace** to replace the search string, and **Find Next** if you do not want to replace a particular entry. To replace all the entries, press **Replace All**. You can replace values of a numeric variable with numbers only.

Find Variable

To search for the variables in the Data Editor, from the menus choose:

- Edit
- Data
 - Find Variable...

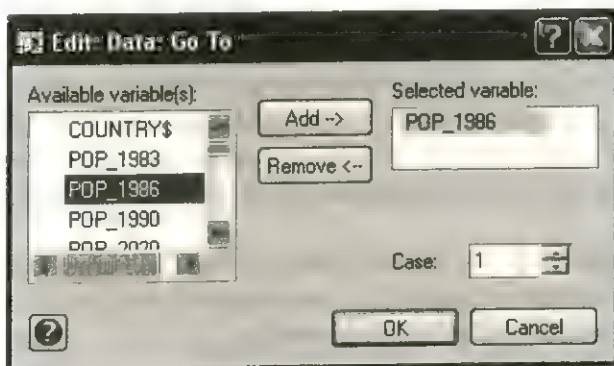


Select a variable from the Available variable(s) list that you want to find.

Go To

To go to a particular case of a variable in the Data Editor, from the menus choose:

Edit
Data
Go To...



The current variable and case number is shown by default. Select a desired variable and specify the case number. The chosen cell will then be highlighted in the Data Editor.

Undo/Redo

The Undo/Redo facility allows you to retrieve any data that may have been modified accidentally. To undo an action in the Data Editor, from the menus, choose

Edit
Undo...

To undo using the keyboard, click in the Data Editor and press Ctrl + Z.

To redo an action in the Data Editor, from the menus, choose

Edit
Redo...

To redo using the keyboard, click in the Data Editor and press Ctrl + Y.

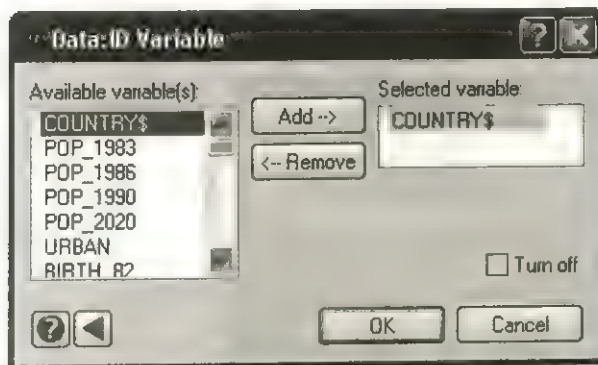
You can undo up to a maximum of 32 recent changes in the data you are currently editing. Any changes you make in the Data Editor can be undone through this facility. For example, manual editing of data, Find and Replace, commands like LET, RANK, SORT, etc.. In the case of variable properties, undo/redo will work only for the changes made to the variable name and type. Also note that undo/redo will not work for any modifications in the data processing conditions, namely, Category, By groups, Select cases, Frequency, Weight, Id variable, and Order of categories.

If Undo/Redo introduces a new variable in the data file, that variable will have exactly the same properties it had when it was removed from the file. For example, suppose you insert a variable, set it to be categorical and then undo the operation. Now, suppose you redo the operation. The variable gets inserted as a categorical variable.

If the variable removed was a frequency variable, Undo will bring it back as a frequency variable provided no frequency setting change has been made in the interim. The same holds for weight and ID variables as well.

Case ID

You can specify an ID variable to be displayed by choosing ID Variable from the Data menu or by using the IDVAR command. The value of the specified variable (instead of the case number) is displayed.



To return to displaying the case number, select Turn off.

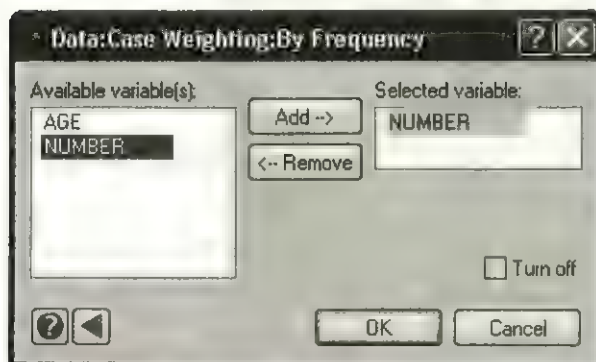
Other Data Structures

In addition to the usual rectangular, case-by-variable organization, SYSTAT accommodates two other data structures:

- Tabulated form.
- Matrix input.

Data in Tabulated Form

Choose By Frequency from Case Weighting in the Data menu or use the FREQ command to identify that the data are counts. That is, case(s) with the same values are entered as a single case with a count. If a variable is declared as a frequency variable, an icon indicating the frequency is displayed on the top of the variable in the Data Editor. Note that frequency works for rectangular data only.



For example, Morrison's data from a breast cancer (*CANCER*) study of 764 women are shown below. Instead of 764 cases, the data file contains 72 records for cells defined by the factors: 1) survival, 2) age group, 3) diagnostic center, and 4) tumor status.

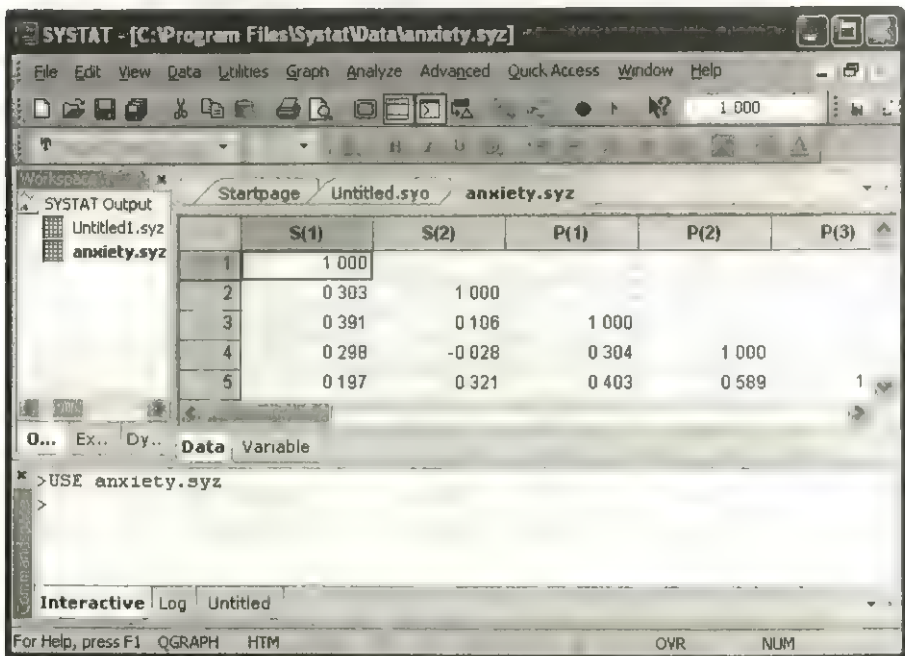
	<i>SURVIVES</i>	<i>AGE</i>	<i>CENTERS</i>	<i>TUMORS</i>	<i>NUMBER</i>
1.	Alive	50	Boston	MaxBengn	0
2.	Alive	50	Boston	MaxMalig	4
3.	Alive	50	Boston	MinBengn	24
4.	Alive	50	Boston	MinMalig	11
5.	Dead	50	Boston	MaxBengn	0
6.	Dead	50	Boston	MaxMalig	6
7.	Dead	50	Boston	MinBengn	7
8.	Dead	50	Boston	MinMalig	6
9.	Alive	50	Glamorgn	MaxBengn	1
10.	Alive	50	Glamorgn	MaxMalig	8

31.	Dead	60	Boston	MinBengn	20
32.	Dead	60	Glamorgn	MinMalig	8
33.	Alive	60	Glamorgn	MaxBengn	4
34.	Alive	60	Glamorgn	MaxMalig	10
35.	Alive	60	Glamorgn	MinBengn	39
36.	Alive	60	Glamorgn	MinMalig	27
37.	Dead	60	Glamorgn	MaxBengn	0
38.	Dead	60	Glamorgn	MaxMalig	3
39.	Dead	60	Glamorgn	MaxBengn	12
40.	Dead	60	Glamorgn	MinMalig	14
		.			
		.			
		.			
63.	Dead	70	Glamorgn	MinBengn	7
64.	Dead	70	Glamorgn	MinMalig	3
65.	Alive	70	Tokyo	MaxBengn	1
66.	Alive	70	Tokyo	MaxMalig	5
67.	Alive	70	Tokyo	MinBengn	6
68.	Alive	70	Tokyo	MinMalig	1
69.	Dead	70	Tokyo	MaxBengn	0
70.	Dead	70	Tokyo	MaxMalig	1
71.	Dead	70	Tokyo	MaxBengn	3
72.	Dead	70	Tokyo	MinMalig	2

NUMBER is the count of women in each cell. Choose **By Frequency** from **Case Weighting** in the **Data** menu to identify this variable as the count variable.

Matrix Input

You can enter a triangular matrix such as a correlation matrix. As usual, variable names go in the top row. Next, type the correlations. Since correlation matrices are symmetric, type only the lower diagonal portion.



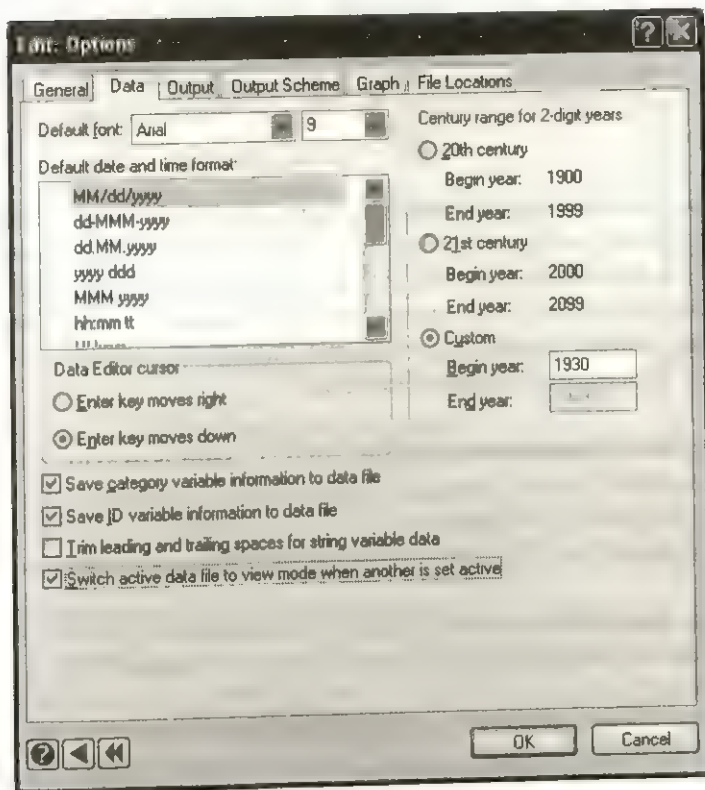
Choose Save As from the File menu to save the data, and from the Save drop-down, click on Save Options to specify the type of matrix. The available types include rectangular (the usual file of case(s) and variable(s)), SSCP, covariance, correlation, dissimilarity, and similarity.

Global Data Options

To customize the behavior of the Data Editor, from the menus choose:

Edit
Options...

Then click the Data tab on the Edit:Options dialog box.



Default font. You can specify the font type and size to use for displaying data values and variable properties in the Data Editor.

Default date and time format. All date and time variables without a specified format appear in this selected format. The number of characters indicates the number of digits or letters displayed.

Data Editor cursor. Controls the navigation behavior of the Enter key in the Data Editor. The Enter key can either move to the next cell on the right for case-by-case entry or down to the next row for variable-by-variable entry.

Set century range for 2-digit years. When dealing with two-digit years, SYSTAT's Data Editor and some date and time function need a reference point for assigning a century. By default, this reference point is 1930 which means all two-digit years are assumed to correspond to the 100-year range from 1930 to 2029. SYSTAT treats two digits years that fall below the final two digits of the Begin year as members of the next century. For example, 28 is treated as 2028 and 31 is treated as 1931. You can select 20th century or 21st century if you want the 100-year range to begin at 1900 or 2000 respectively.

Save category variable information to data file. By default, if you define a variable to be categorical and save the data file subsequently, the category information is saved to the file. This implies that you need not declare the variable to be categorical every time you open the data file. However, you may want to uncheck this option if you need to process the variable as non-categorical sometimes, or if you need to process different sets of categorical variables each time.

Save ID variable information to data file. An ID variable is used to identify cases in the output, instead of simply displaying case numbers. By default, like categorical information, ID variable information is also saved to the data file. You can uncheck this option if you do not want ID variable values to identify cases all the time.

Trim leading and trailing spaces for string variable data. SYSTAT allows you to enter string data values with leading and trailing spaces. Some of SYSTAT's character functions, namely LFT\$, CNT\$, RGT\$, LPD\$ and RPD\$, also insert leading and/or trailing spaces in string values. Check this option if you want such spaces to be trimmed out for processing by the LABEL, ORDER and RECODE commands.

Switch active data file to view mode when another is set active. By default, when a data file is opened, the currently active data file closes down after prompting the user to save changes, if any. In such situations, you can set the currently active data file to switch to view mode instead of closing down.

Using Commands

To define variable properties for numeric variables, use:

```
DEFVAR varlist / TYPE = NUMBER or DATE or EXPONENTIAL,
                DISPLAY = m.n or 'dateformat', COMMENT = 'text'

VARLAB varlist/'label'

LABEL varlist/n1='text1',n2='text2',...
                                     or
                n1,n2,...='text1',
                n3,n4,...='text2',
                                     or
                n1..n2 = 'text1',
                n3..n4 = 'text2',...
```

To define variable properties for string variables, use:

```
DEFVAR varlist$ / TYPE=STRING DISPLAY = n COMMENT = 'text'

VARLAB varlist$/'label'

LABEL varlist$/'oldtext1'= newtext1, 'oldtext2'= newtext2,...
```

You may use VALLAB instead of LABEL.

To control the display of variable labels, use:

```
VDISPLAY LABELS or NAME or BOTH
```

To control the display of value labels, use:

```
LDISPLAY LABELS or DATA or BOTH
```

To specify category variable(s), use:

```
CATEGORY varlist
```

To reset the previously declared category variables in *varlist*, use:

```
CATEGORY varlist/OFF
```

Some more options of CATEGORY are available. Refer the *Language Reference* for details.

To specify an ID variable:

IDVAR *var*

Typing IDVAR without an argument returns the display to case numbers.

The following editing commands are also available:

REPEAT <i>n</i>	Adds <i>n</i> number of empty cases to a new data file. If the data file already contains <i>m</i> cases and $m < n$, then adds $n - m$ cases. If $m \geq n$, then no cases are added.
INSERT <i>nloc</i> , <i>m</i> / ROWS or COLUMNS	Inserts <i>m</i> blank rows or columns at location <i>nloc</i> .
CUT <i>var1</i> or <i>varlist</i>	Removes variables from the data file to the clipboard.
PASTE <i>nloc</i>	Pastes clipboard contents into the file at location <i>nloc</i> (a row or column number, depending on the contents of the clipboard).
DELETE <i>n1</i> , <i>n2</i> , <i>n3</i> , ...	Deletes specific cases from the data file to the clipboard.

Data Transformations

Leland Wilkinson and Laszlo Engelman

(revised by Mangalmurti Badgujar, Anand Ramchandran, and S. Anoopama)

Often, the data as recorded are not sufficient for a complete analysis. For example, you may need to reexpress values to improve symmetry, derive new variables such as a total score or a ratio of two variables, select a subset of the cases, etc. To execute these tasks, you specify statements in this form:

```
LET variable = expression  
IF condition THEN LET assignment  
SELECT condition1 AND condition2 OR ...
```

SYSTAT provides a variety of functions that you can use in these statements. You can:

Derive new variables. If you have two quiz scores and a final exam score for each student, you can use the arithmetic operators + and * to compute their total score:

```
LET total = quiz1 + quiz2 + 2*final
```

Reexpress data. Transformed values may meet assumptions required for statistical analyses better than the original data. For example, often the distribution of the logarithm of body weight is more symmetric than that of untransformed weights. To reexpress weight in log base 10 units, use the L10 function:

```
LET lg_wt = L10(weight)
```

Compute summary variables. For each case, SYSTAT's multivariable functions compute results across variables. For example, use the AVG function to compute the average of the scores S1 to S6 for each subject:

```
LET average = AVG(s1,s2,s3,s4,s5,s6)
```


Assign new values conditionally. For example, using the variable *TEMP* (winter temperature), derive a new variable *TEMP\$* containing a character descriptor:

```
IF temp < 32 THEN LET temp$ = 'Cold'
```

Execute global edits. Use relational operators (<, >, and =) and logical operators (AND and OR) in conditional transformations. For example, the statement

```
IF (age > 20 AND income < 7000 AND job$ = 'No'),  
  THEN LET relief$ = 'Yes'
```

stores the word *Yes* in the variable named *RELIEF\$* for every subject over 20 years old who reports an income of less than \$7,000 and is unemployed.

Select cases. Use relational operators to restrict your analysis to subjects satisfying certain conditions. For example,

```
SELECT age >= 21
```

omits all subjects under 21.

Rank, center or standardize the values of a variable. You can rank the values of a variable from smallest to largest, center the values around their mean, or standardize the values of a variable with their *z* scores or score on a 0,1 scale.

Manipulate character variables. Use SYSTAT's character functions to manipulate character values, including changing case, extracting characters or text strings, converting characters to numbers, etc.

Format and transform dates and times. For example, you could write the fifth of February, 2007, as Feb 5, 2007, 02/05/07, or 05-02-07; or, you could compute the time between events in, say, days or even seconds.

Generate random numbers. SYSTAT has functions to generate values from a variety of univariate discrete and continuous distributions. In addition, density, cumulative and inverse cumulative functions are available for univariate continuous and discrete distributions. For more details, refer to the section "Functions Relating to Probability Distributions" on page 96.

For more complex data transformations, you can use IF...THEN... ELSE statements, FOR...NEXT or WHILE...ENDWHILE loops, ARRAY, and DIM (Dimension) statements.

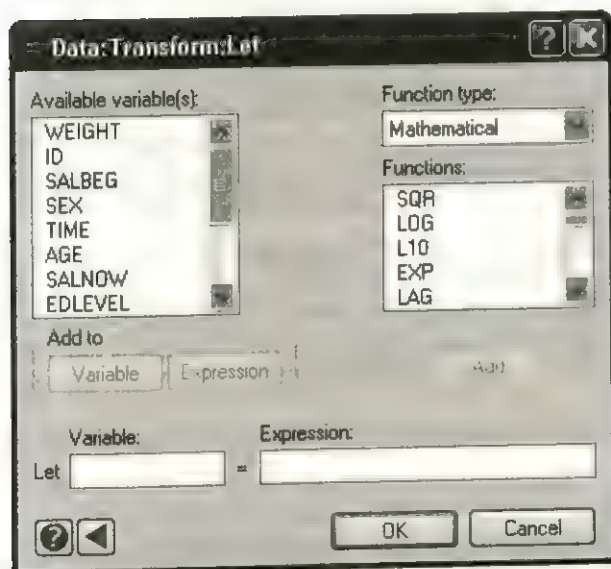
Let Dialog Box

LET assigns values to a variable according to a specified expression. You can use any mathematically valid combination of variables, numbers, functions, and operators, including complex expressions joined by the logical operators AND, OR, or NOT. Valid expressions include:

```
LET CHANGE = WEEK2 - WEEK1
LET LOGIT  = 1 / (1 + EXP(A+B*X))
LET TRENDY = (INCOME > 60000) AND (CAR$ = 'BMW')
```

To open the Let dialog box, from the menus choose:

Data
Transform
Let...



To specify the target variable, select a variable from the Available variable(s) list and press the Variable button. If you want to create a new variable, simply type it into the box. Select the function type, functions and variable(s) to be used in the expression from the list and use the Add and Expression buttons to add them to the expression. Your expression will appear in the Expression area as you create it.

You can transform existing variables or create new ones. If the target variable (the variable following LET) already exists in the file, its values are replaced by the value of the expression for each case. If the variable does not exist, then a new variable with that name is added in the first column at the right side of the data matrix. For example, LET B = A creates a new variable named *B* if *B* does not already exist. If it does exist, its values are replaced by the values of *A*.

Rules For Expressions

- If the expression contains any character values, they must be enclosed in single or double quotation marks. Character values are case sensitive.
- Arguments for functions must be inside parentheses, for example, LOG(weight) or SQR(income). If there are two or more arguments, they must be separated by commas.

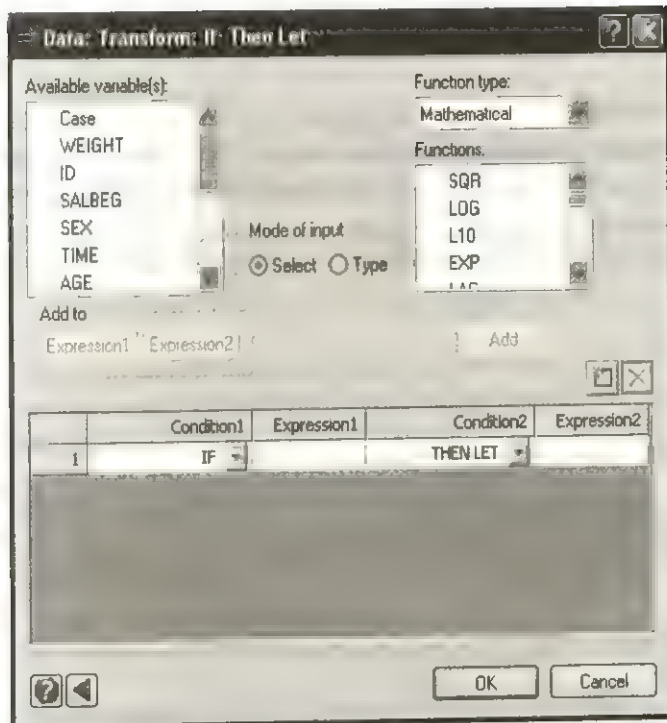
If...Then Let Dialog Box

IF...THEN LET allows you to transform variables conditionally. For all cases where the condition is true, SYSTAT executes the action. You can use any mathematically valid combination of variables, numbers, functions, and operators. Examples of valid statements are:

```
IF AGE > 80 THEN LET AGE$ = 'ELDERLY'  
IF X = 99 THEN LET X = .  
IF (SEX$ = 'MALE') AND (AGE > 30) THEN LET GROUP = 1
```

To open the If...Then Let dialog box, from the menus choose:

Data
Transform
If...Then Let...



You can transform existing variables or create new ones using the If Then Let dialog. If the target variable (the variable following *LET*) already exists in the file, its values are replaced by the transformation value for each case. If the target variable does not exist, then a new variable with that name is added in the first column at the right side of the data matrix.

Mode of input. Select one of the following two options:

- **Select.** Select the variables and functions to build the transformation expression. To add an element to the transformation expression, position the cursor in the desired location before selecting the variable or function to be added. Choose the appropriate condition from the Condition box. To add a variable to the expression, click on it and click the Expression button. Choose the appropriate Function type and function from the Functions box and click the Add button. If you want to create a new variable, simply type it into the desired Expression box.
- **Type.** Type the variables and functions to get the desired transformation expression. To add a variable from the list, use the Add button. Choose the

appropriate Function type and function from the Functions box and click the Add button.

Rules For Expressions

- If the expression contains any character values, those values must be enclosed in single or double quotation marks. Character values are case sensitive.
- Arguments for functions must be inside parentheses, for example, LOG(weight) or SQR(income). If there are two or more arguments, they must be separated by commas.

Built-In Variables

Built-in variables allow you to index aspects of files:

CASE	Case (observation) number
COMPLETE	Column of 1's (for complete cases) and 0's (for cases containing missing values)

For example, if your cases are ordered by time, create a new variable named *TIME* containing the case sequence numbers:

```
LET time = CASE
```

Shortcuts for Transformations

There are several shortcuts you can use to minimize typing for transformation statements.

Multiple transformations: The @ sign. To transform several variables, you can specify a LET statement for each:

```
LET gdp_cap = L10(gdp_cap)
LET mil = L10(mil)
LET gnp_86 = L10(gnp_86)
```

Or, you can use the @ sign to specify the same transformations in one statement:

```
LET (gdp_cap, mil, gnp_86) = L10(@)
```


The syntax for multiple transformations is:

```
LET (var1, var2, var3,...) = expression with @ sign
```

The variable names must be separated by commas or space and enclosed within parentheses (). The @ sign is a placeholder for the variable names. The function can be any valid SYSTAT function or expression.

This @ shortcut notation can also be used with relational operators:

```
LET (blue,depress,cry,sad,no_eat,getgoing) = @ <> 0
```

This dichotomizes the values of six variables into 0 and 1.0. When the value of a variable is not 0, the statement on the right is true, so the result is 1.0. If the value is 0, the statement is false, so the result is 0.0.

Specifying contiguous variables. As a shortcut when selecting adjacent variables in a file, type the first and last variable names and use a double period (..) to include all variables in between:

```
LET total = SUM(score1, score2, score3, score4)
LET total = SUM(score1 .. score4)
```

This shortcut can also be used with the @ sign shortcut:

```
LET (blue .. getgoing) = @ <> 0
```

When using subscripts (such as *Q(1)*, *Q(2)*, etc.), variables that are not contiguous can be specified using the double-period shortcut. For example,

```
STANDARDIZE Q(1) .. Q(10)
```

standardizes the variables *Q(1)* to *Q(10)*, regardless of their order in the data file.

Operators and Functions

SYSTAT's functions and operators can be used:

- In any LET, IF...THEN LET, or SELECT expression.
- In some FIND expressions (depending on data type).
- In the calculator.

Operators are categorized as arithmetic, relational, and logical. Functions are categorized as mathematical, multivariable, multibase, group and interval, date and time, character, and distribution.

Arithmetic Operators

You can use the following arithmetic operators:

+	Addition	*	Multiplication
-	Subtraction or negative sign	/	Division
** or ^	Exponentiation		

Some examples include:

```
quiz1 + quiz2
y - 2
year*year
birth_rt / death_rt
x^2
```

These could be used, for example, as follows:

```
LET total = quiz1 + quiz2
IF (city$ = 'LA') THEN LET code=y-2
```

SYSTAT sets the resulting values of inadmissible operations (for example, division by 0) to missing.

Relational Operators

Relational operators are used in relational expressions to compare either two numeric or two character expressions.

= or ==	Equal to	<>	Not equal to
<	Less than	>	Greater than
<=	Less than or equal to	>=	Greater than or equal to

Some examples include:

```
score1 < score2
state$ = 'NY'
age > 21
x <> .
age <= 65
income >= 3000
```

These could be used, for example, as follows:

```
IF (state$ == 'NY') THEN LET region = 1
```

Logical Operators

The logical operators AND and OR connect two relational expressions, and NOT negates a logical expression. These operators yield results that are either true or false.

This expression selects people of work-force age (those older than 17 and younger than or equal to 65):

```
age > 17 AND age <= 65
```

Note that even if you are testing for different conditions on the same variable, you must specify a complete relational expression for each condition.

Correct: age > 17 AND age <= 65

Incorrect: age > 17 AND <= 65

Here, we select people who are not of work-force age (those younger than or equal to age 17 or older than age 65):

```
age <= 17 OR age > 65
```

The logical operator NOT reverses the value of an entire expression. To select people who are not of work-force age, we could also specify:

```
NOT (age > 17 AND age <= 65)
```

If the result of an expression is true, it is assigned a value of 1; if it is false, it is assigned a value of 0. NOT sets any nonzero value (true) to 0, and any 0 value (false) to 1 (true).

SYSTAT follows the standard for programming languages and returns 0 for false and 1 for true.

Logical AND

The following example demonstrates how a result is determined:

```
LET trendy = income > 40000 AND car$=='BMW'
```

For any case where the subject's income is over \$40,000 AND the subject's car is a BMW, the result of the expression is true and the value of *TRENDY* is 1. If the subject has a lower income OR a different car, the result is false and *TRENDY* is assigned 0. If the subject fails to report income or car model, the value of *TRENDY* is set to missing.

In summary, when AND is used:

Condition 1		Condition 2		Result	Value
True	AND	True	yields	True	1
True	AND	False	yields	False	0
False	AND	True	yields	False	0
False	AND	False	yields	False	0
True	AND	Missing	yields	Missing	.
Missing	AND	True	yields	Missing	.
False	AND	Missing	yields	False	0
Missing	AND	False	yields	False	0
Missing	AND	Missing	yields	Missing	.

Logical OR

If the logical operator OR is used instead of AND, for example:

```
LET TRENDY = INCOME > 40000 OR CAR$=='BMW'
```


only one of the conditions has to be true for the result to be true (1).

<i>Condition 1</i>		<i>Condition 2</i>		<i>Result</i>	<i>Value</i>
True	OR	True	yields	True	1
True	OR	False	yields	True	1
False	OR	True	yields	True	1
False	OR	False	yields	False	0
True	OR	Missing	yields	True	1
Missing	OR	True	yields	True	1
False	OR	Missing	yields	Missing	.
Missing	OR	False	yields	Missing	.
Missing	OR	Missing	yields	Missing	.

Order of Expression Evaluation

Expressions are evaluated from left to right according to the precedence of operators. That is, operators with higher precedence are evaluated before those with lower precedence. Order of precedence from highest to lowest runs as follows:

1. Expressions enclosed in parentheses $()$
2. Exponentiation $^$ or $**$
3. Negation $-$
4. Multiplication and division $*, /$
5. Addition and subtraction $+, -$
6. Comparison $=, <, >, <=, >=$
7. Logical negation NOT
8. Logical comparison AND
9. Logical comparison OR

Because of this order of precedence, the expressions $(A + B \times C)$ and $(A + (B \times C))$ are the same, but they differ from $((A + B) \times C)$.

Multiplication has a higher precedence than addition and is therefore performed first. If $A = 1$, $B = 2$, and $C = 3$, the above expressions evaluate as follows:

$$A + B \times C = 1 + 2 \times 3 = 1 + 6 = 7$$

$$A + (B \times C) = 1 + (2 \times 3) = 1 + 6 = 7$$

$$(A + B) \times C = (1 + 2) \times 3 = 3 \times 3 = 9$$

The only exception to the “left to right” rule is with exponentiation. The exponentiation $a^b{}^c$ is evaluated “right to left”; that is, $a^b{}^c$ means $a^{(b^c)}$, or a^{b^c} .

Mathematical Functions

These functions modify the variable, number or expression that you place inside the parentheses. Some examples include:

<i>Function</i>	<i>Result</i>
SQR (a)	Square root of a [$a \geq 0$].
LOG (a)	Natural logarithm of a [$a > 0$].
L10 (a)	Logarithm base 10 [$a > 0$].
EXP (a)	Exponential function e^a .
LAG (var, n)	Lag variable by shifting values down n rows; if n is omitted, default = 1.0.
INT(a)	Integer part of a .
LGM (a)	Log gamma: $LGM(n) = \log(\Gamma(n))$, [$=\log((n-1)!) \text{ when } n \text{ is a positive integer}$]
SGN (a)	-1 if $a < 0$, 0 if $a = 0$, and 1 if $a > 0$.
ABS (a)	Absolute value $ a $.
SIN (a)	Sine of a (in radians).
COS (a)	Cosine of a (in radians).
TAN (a)	Tangent of a (in radians).
ASN (a)	Arcsine of a (which yields radian results).
ACS (a)	Arccosine of a (which yields radian results).
TNH (a)	Hyperbolic tangent of a .
ATN (a)	Arctangent of a (which yields radian results).
ATH (a)	Arc hyperbolic tangent of a (Fisher's Z , which yields radian results).
AT2 (a,b)	Arctangent with sine (a) and cosine (b) argument.
MOD (a,b)	The remainder of a/b .

```
LOG(weight)
SQR(3)
INT(cardio+cancer)
SIN(COS(TAN(3*Y)))
```

These could be used, for example, as follows:

```
LET lg_weight = LOG(weight)
```

Arguments of trigonometric functions must be in radians, not degrees. LOG is the natural logarithm and L10 is the base 10 logarithm. You can obtain logs in other bases by dividing the natural log by the log of the base; for example, for $\log_2 A$, use $\text{LOG}(A)/\text{LOG}(2)$. SYSTAT sets the resulting values of inadmissible operations (for example, square roots of negative numbers) to missing.

Multivariable Functions

The functions described below operate on values of variables within a case. With these functions, you can:

- Compute summary descriptive statistics across values of selected variables.
- Search selected variables for specific numeric or character values.
- Compute the slope of the line of best fit through points with equally spaced x values or values that you specify.
- Compute the area under a curve created by connecting equally spaced points or points with spacing that you specify.

Function	Result
MIS(y_1, y_2, \dots)	Number of missing values.
NUM(y_1, y_2, \dots)	Number of values that are not missing (number of usable values).
AVG(y_1, y_2, \dots)	Mean of nonmissing values.
STD(y_1, y_2, \dots)	Standard deviation of nonmissing values.
MIN(y_1, y_2, \dots)	Smallest among nonmissing values (minimum value).
MAX(y_1, y_2, \dots)	Largest among nonmissing values (maximum value).
SUM(y_1, y_2, \dots)	Sum of nonmissing values.
SLE(y_1, y_2, \dots)	Coefficient b of the regression line $y = a + bx$, where x 's are assumed to be equally spaced at a distance of 1 unit.
SLU($x_1, y_1, x_2, y_2, \dots$)	Coefficient b of the regression line $y = a + bx$.
ARE(y_1, y_2, \dots)	Area under the values of (x_i, y_i) , where x 's are assumed to be equally spaced at a distance of 1 unit.
ARU($x_1, y_1, x_2, y_2, \dots$)	Area under y by the trapezoidal rule.
COD($number, var1, var2, \dots$)	The index (using integers 1, 2, 3, ...) when $number$ matches a value in $var1$ through $varp$, respectively; 0 otherwise.
COD('charval', $var1$, var2$, ...)$	The index (using integers 1, 2, 3, ...) of $var1$, var2$, ... when charval matches the value of the respective variable; 0 otherwise.$
INC($number, var1, var2, \dots$)	A 1 (true) if $number$ matches a value in $var1, var2, \dots$, or $varn$; 0 (false) otherwise.
INC('charval', $var1$, var2$, ...)$	A 1 (true) when $charval$ matches a value in $var1$, var2$, ..., or varn$; 0 (false) otherwise.$

- Arguments are not restricted to variable names—they can be explicit values or other functions—and they can contain arithmetic operators, for example, (INDEX-1).

- Arguments of a function must be separated by commas. For example, when using AVG to compute the mean, specify: AVG(10,313,-29,236,19).
- Use a double period (..) as a shortcut notation to shorten a list of contiguous variables—that is, for Q1, Q2, ..., Q20, use Q1 .. Q20.

Multicase Functions

The functions described below operate on cases of a variable. With these functions you can compute a value for specified cases.

Function	Result
CMIN(<i>variable, initial case, end case, step</i>)	Smallest among non-missing values (minimum value).
CMAx(<i>variable, initial case, end case, step</i>)	Largest among non-missing values (maximum value).
CSUM(<i>variable, initial case, end case, step</i>)	Sum of non-missing values.
CPROD(<i>variable, initial case, end case, step</i>)	Product of non-missing values.
CRANGE(<i>variable, initial case, end case, step</i>)	Range of non-missing values (maximum value - minimum value).
CMEAN(<i>variable, initial case, end case, step</i>)	Mean of non-missing values.
CVAR(<i>variable, initial case, end case, step</i>)	Variance of non-missing values.
CSTD(<i>variable, initial case, end case, step</i>)	Standard deviation of non-missing values.

- Initial case and step are '1' by default.
- Default end case is the last case of the file in use.
- Argument may not be just a variable name - it can be a mathematical function, containing the variable as an argument, and it can contain arithmetic operators. Here is an example of a data file:

X	Y	Z
1	6	11
2	7	12
3	8	13
4	9	14
5	10	15

```
EXP (CSUM (Y* (X+Z) ) )
```


Multiplies the sum of X and Z, with Y for each case and adds up the evaluated values. The exponential of the resultant is 4.308817E+286.

Group and Interval Functions

The COD function is used to recode values of a grouping variable. The INC function compares each value of a numeric or character variable with a list of values you specify. If a match is found, SYSTAT returns a 1; if not, SYSTAT returns a 0. The CUT function lets you define intervals on a quantitative (continuous) variable or divide the values of a character variable alphabetically.

Function	Result
COD(var,num1,num2,...)	Values of <i>var</i> are replaced: <i>num1</i> becomes 1, <i>num2</i> becomes 2, etc.
COD(var\$,'text1','text2',...)	Values of <i>var\$</i> are replaced: <i>text1</i> becomes 1, <i>text2</i> becomes 2, etc.
INC(var,num1,num2,...)	A 1 (true) if the value in <i>var</i> matches one of the numbers <i>num1</i> , <i>num2</i> , ...; 0 (false) otherwise.
INC(var\$,'text1','text2',...)	A 1 (true) if the value in <i>var\$</i> matches a value in <i>text1</i> , <i>text2</i> , ...; 0 (false) otherwise.
CUT(var,num1,num2,...)	Use to define intervals along a continuous variable. Values in <i>var</i> less than or equal to <i>num1</i> get value 1. Values greater than <i>num1</i> and less than or equal to <i>num2</i> get value 2, etc.
CUT(var\$,'text1','text2',...)	Use to group variables alphabetically. Values in <i>var\$</i> less than <i>text1</i> (alphabetically), including <i>text1</i> , get value 1. Values between <i>text1</i> and <i>text2</i> , including <i>text2</i> , get value 2, etc.

Date and Time Functions

SYSTAT has functions that format and transform dates and times. SYSTAT can:

- Give the current date or time in the format you specify.
- Transform date or time character values to numeric values.
- Transform numeric date and time values to character values with a specified format.
- Return the day of the century (DOC) that corresponds to a given year, month, and day. Use DOC to compute time between events.
- Return the day of the week corresponding to a given day of the century.

- Use the following functions to change the format of dates and times, to convert numbers to dates and times, and vice versa.

Function	Result
NOW\$(<i>'format'</i>)	The current time and/or date in format you specify.
VAL(<i>var\$</i> , <i>'format'</i> , <i>field</i>)	Extract numbers from dates or times with character values in <i>var\$</i> . <i>Format</i> is the format of the dates or times in <i>var\$</i> . <i>Field</i> is the number of the <i>format</i> element you want to extract (for example, for <i>format</i> mm/dd/yy, 1 corresponds to month, 2 to day, and 3 to year).
STR\$(<i>var</i> , <i>'format'</i>)	Write numeric dates or time values stored in <i>var</i> as characters using <i>format</i> .
DOC(<i>var\$</i> , <i>'format'</i>)	Return day of century from a character variable containing a date in the specified format.
DOC(<i>yvar</i> , <i>mvar</i> , <i>dvar</i>)	Return day of century from year, month, and day variables (specify the arguments in this order).
DOW\$(<i>docvar</i>)	Return day of week (Monday, Tuesday, ...) from numeric day of century <i>docvar</i> .
DAT(<i>n</i> , <i>'format'</i>)	Return day or time from a numeric day of century <i>n</i> in specified format of either Y (year), M (month), D (day), h (hour), m (minute), or s (second)—specify only one.

In these date and time functions, SYSTAT uses the following built-in symbols:

D	Day	s	Second	.	Decimal point
M	Month	m	Minute	#	A number before or after a decimal
Y	Year	h	Hour		

Note that hours, minutes, and seconds are assumed to be coded as fractions of days. Also, the number of *m*'s indicates how the month is reported:

MM	1 through 12
MMM	Jan, Feb, Mar, Apr, May, ..., Dec
MMMMMMMM	January, February, March, April, May, ..., December

Current Time and Date: NOW\$

The NOW\$ function gives the current date or time. You can use NOW\$ in the calculator or as part of a transformation statement. For example, if on December 25, 2006, you type

```
CALC NOW$ ( 'mm/dd/YY' )
```


SYSTAT informs you that it is:

12/25/06

If you type

```
CALC NOW$('mmm dd, yyyy')
```

the result is:

Dec 25, 2006

The following are the same NOW\$ functions used in LET statements:

```
LET TIME$ = NOW$('hh:mm:ss')  
LET DATE1$ = NOW$('mm/dd/yy')  
LET DATE2$ = NOW$('mmm dd, yyyy')
```

Converting Times and Dates to Numbers: VAL

The VAL function converts date or time character values to numeric values. You must specify the format of the character variable and indicate which field you want to save as a numeric variable. For example,

```
LET month = VAL(date1$, 'mm/dd/yy', 1)
```

- In this command, *DATE\$* is the name of the character variable holding the dates.
- mm/dd/yy tells SYSTAT that the first two characters of each *DATE\$* value represent the month. This is followed by a slash, two characters representing the day of the month, another slash, and two characters representing the year. For example, a value of *DATE\$* could be 03/24/04.
- The last parameter tells SYSTAT which field to read into the new variable. The 1 in this example represents the month field. So, for the date 03/24/04, SYSTAT reads the value 3 and stores it in the numeric variable *MONTH*.

Converting Times or Dates to Characters: STR\$

The STR\$ function transforms a numeric date or time to a character value using the format you specify:

```
LET newvar$ = STR$(oldvar, 'format')
```


SYSTAT interprets the values as days.

The LET statement,

```
LET date$ = STR$(dayofcnt, 'mm/dd/yyyy')
```

transforms the following day-of-the-century values in *DAYOFCNT* to the character dates in *DATE\$*:

<i>DAYOFCNT</i>	<i>DATES</i>
1	1/1/1900
366	1/1/1901
15316	12/7/1941
30000	2/19/1982
38000	1/15/2004

Day of the Century: DOC

The DOC function gives the day of the century corresponding to a particular year, month, and day. SYSTAT begins counting at December 31, 1899; DOC returns a value of 0 for this date. Dates before this zero point appear as negative values. Dates after December 31, 1899 yield the number of days between the zero point and the date in question. Thus, DOC returns a value of 36525 for January 1, 2000, rather than returning a value of 1. The syntax of the DOC function is

```
DOC(year, month, day)
DOC(yvar, mvar, dvar) or DOC(var$, 'format')
```

where *year* is either a year value or a variable containing year values, *month* is a month value or a variable containing month values, and *day* is a day of the month value or a variable containing the day of the month. A month value must be the number of the month (for example, 12) rather than the name of the month (December). A year value can be a complete year (for example, 1964) or just the last two digits (64). Use the Data tab of the Options dialog in the Edit menu (or the WINDOW command) to specify the century to use for two digit years.

Get the Day of the Week: DOW\$

The DOW\$ function gives the day of the week corresponding to a given day of the century. The syntax of the DOW\$ function is

```
DOW$(dayofcnt)
```

where dayofcnt is either a day-of-the-century value or a variable containing the day-of-the-century value.

You can embed the DOC function in the DOW\$ function to find the day of the week corresponding to a particular date. For example, to find out on which day of the week October 24, 1945, fell, type:

```
CALC DOW$(DOC(1945,10,24))
```

SYSTAT responds:

```
Wednesday
```

Character Functions

SYSTAT has the following functions for manipulating character values:

UPR\$	Change to caps.	IND	Find the position of a character.
LOW\$	Change to lower case.	SND\$	Soundex encoding.
CAP\$	Initial caps.	CAT\$	Join character strings.
CNT\$	Center.	MID\$	Extract characters.
RGT\$	Right-justify.	SUB\$	Replace text1 with text2.
LFT\$	Left-justify.	PUT\$	Replace characters.
SQZ\$	Remove embedded blanks.	RPD\$	Right-pad with character string.
SQZ\$	Remove text.	LPD\$	Left-pad with character string.
VAL	Convert character numbers to numeric values.	ASC\$	ASCII characters.
STR\$	Convert numbers to character values.	ICH	ASCII codes.
LAB\$	Extract values specified by LABEL.		

These functions can:

- Convert the values of a character variable to all upper case or all lower case, or capitalize only the first letter and make the rest of the letters lower case.
- Center, or left- or right-justify character values.
- Delete blanks or specified text from character values.
- Find the position of a particular character in each value of a character variable.
- Convert numbers to character values or character variables containing numbers to numeric variables.
- Extract text strings from or insert text strings into character values.
- Use Soundex coding to create four-digit, alphanumeric codes based on the sound of the word rather than its spelling.
- Concatenate the values of two character variables into a single variable.
- Find the ASCII code corresponding to a character or find the character corresponding to the ASCII code.

In this version of SYSTAT, results of operations on character variables are limited to 256 characters, or leading characters relevant to the function.

Changing Case: UPR\$, LOW\$, and CAP\$

The UPR\$, LOW\$, and CAP\$ functions allow you to change the case of character values.

<i>Function</i>	<i>Result</i>
UPR\$(var\$)	Change the values of var\$ to all capitals.
LOW\$(var\$)	Change the values of var\$ to all lower case.
CAP\$(var\$)	Capitalize the first letter of each var\$ value. Make the remaining letters of each value lower case.

Justifying Character Values: CNT\$, RGT\$, and LFT\$

The CNT\$, RGT\$, and LFT\$ functions center and justify character values.

<i>Function</i>	<i>Result</i>
CNT\$(var\$,n)	Center values of variable <i>var\$</i> , where <i>n</i> is width of character field. If you do not specify <i>n</i> , SYSTAT assumes a width of 72.
CNT\$('text')	Center text in a 72-character field.
RGT\$(var\$,n)	Right-justify values of variable <i>var\$</i> , where <i>n</i> is width of character field. If you do not specify <i>n</i> , SYSTAT assumes a width of 72.
RGT\$('text')	Right-justify text in a 72-character field.
LFT\$(var\$)	Left-justify values of <i>var\$</i> .
LFT\$('text')	Left-justify text.

SYSTAT assumes that each character field is 72 characters wide, which centers text on the screen or page. Internally, transformations use 72 characters.

Deleting Characters: SQZ\$

Use SQZ\$ to delete blanks, commas, apostrophes, or other characters from character values.

<i>Function</i>	<i>Result</i>
SQZ\$(var\$)	Remove all embedded blanks.
SQZ\$(var\$, 'text')	Remove text from the values of <i>var\$</i> .

Locating a Character in a String: IND

The IND function locates a particular character in each value of a character variable.

<i>Function</i>	<i>Result</i>
IND(var\$, 'char')	Find the position of the first occurrence of <i>char</i> in each value of <i>var\$</i> .

For example, the following LET statement:

```
LET position = IND(team$, 'e')
```


finds the character position of the letter *e* in each team name. The *POSITION* values that SYSTAT returns for a given value of *TEAM\$* are:

<i>TEAMS</i>	<i>POSITION</i>
Mets	2
Yankees	5
Blue Jays	4
Cubs	0

Notice that SYSTAT returns the position of the first *e* in *Yankees*.

Converting Numbers to Characters and Vice Versa: VAL and STR\$

Although SYSTAT allows you to type numbers as values of character variables, the numbers are considered characters and cannot be used in calculations. You can use the VAL function to convert the values into numbers. Conversely, you can use the STR\$ function to convert numbers to characters.

<i>Function</i>	<i>Result</i>
VAL(<i>var\$</i>)	Convert numbers stored in the character variable <i>var\$</i> to numeric values that can be used in calculations.
STR\$(<i>var</i>)	Convert numeric values of <i>var</i> to character values.

Suppose you entered years as numbers but want to use the values to label points in a plot. First, convert the numbers to characters and store them in the variable *YEAR\$*:

```
LET year$ = STR$(year)
```

Then specify *YEAR\$* as the label. For example:

```
PLOT yvar*xvar / LABEL=year$
```

Now suppose, conversely, that you enter the year as characters and then want to do calculations. Use the VAL function to convert the character values to numbers:

```
LET year = VAL(year$)
```


Extracting and Inserting Characters

SYSTAT has several functions for manipulating character strings.

<i>Function</i>	<i>Result</i>
MID\$(var\$,p,j)	Extract a string of <i>j</i> characters from each value of <i>var\$</i> , beginning with the <i>p</i> th character.
SUB\$(var\$, 'text1', 'text2')	Replace <i>text1</i> with <i>text2</i> .
PUT\$(var\$, 'text', p, j)	Beginning at the <i>p</i> th character of <i>var\$</i> , replace the next <i>j</i> characters with the first <i>j</i> characters of <i>text</i> .
RPD\$(var\$, 'char')	Right-pad the values of <i>var\$</i> with <i>char</i> .
LPD\$(var\$, 'char')	Left-pad the values of <i>var\$</i> with <i>char</i> .

Concatenating Character Strings: CAT\$

Use the CAT\$ function to join the values of two variables into a single variable.

<i>Function</i>	<i>Result</i>
CAT\$(var1\$, var2\$)	Join contents of <i>var1\$</i> with <i>var2\$</i> , eliminating trailing blanks.

In this version of SYSTAT, the result is limited to 256 characters.

ASCII Codes: ICH

The ICH function gives the ASCII code for a character.

<i>Function</i>	<i>Result</i>
ICH('char')	ASCII code corresponding to the character <i>char</i> .
ICH(var\$)	ASCII code corresponding to the value in <i>var\$</i> .
ASC\$(int)	ASCII character corresponding to the integer <i>int</i> .
ASC\$(var)	ASCII character corresponding to the value in <i>var</i> .

The following table shows some characters and their ASCII codes:

<i>Character</i>	<i>ASCII code</i>
A	65
a	97
3	51
?	63
#	35
}	125

So, for example, ICH('3')=51 and ASC\$(35)=#.

To obtain a table of ASCII codes, specify:

```
NEW
REPEAT 1
FOR I = 1 to 256
A$ = ASC$(I)
PRINT I, A$
NEXT
```

Using Soundex to Code Names: SND\$

The Soundex encoding function (SND\$) allows you to create an alphanumeric code of a word based on how the word sounds rather than its spelling. The SND\$ function can be used to create a new variable to select similarly spelled duplicate records.

<i>Function</i>	<i>Result</i>
SND\$(var\$)	An alphanumeric Soundex code of var\$.

The SND\$ function uses a formula to replace letters with numbers in an *Xnnn* format, where *X* is the letter with which the word begins and *nnn* is up to three numbers indicating the “sound” of the rest of the word.

Soundex ignores vowels (*A, E, I, O, U*) and the consonants *H, W, and Y*. It then codes for the remaining letters:

B, F P, V	1
C, G, J, K, Q, S, X, Z	2
D, T	3
L	4
M, N	5
R	6

If the same code occurs consecutively, the second occurrence is skipped. Some names with their Soundex codes are:

Berry	B6
Wilson	W425
Anderson	A536
Johnson	J525
Smith	S53
Carlson	C642

For a quick way to see the Soundex code of a name, use SYSTAT's calculator:

```
CALC SND$( 'Smith' )
```

SYSTAT returns:

```
S53
```

Functions Relating to Probability Distributions

An important aspect of statistical analysis involves the calculation of values of the **probability mass function (pmf)**, **probability density function (pdf)**, **cumulative distribution function (cdf)**, and **inverse of cdf** (cut-off points) relating to various standard distributions and drawing random samples from them. SYSTAT computes these functions called by \$CF, \$DF, \$IF and \$RN respectively with '\$' denoting a symbol(s) for the distribution (e.g., Z for normal, T for Student's t, DE for double-exponential etc.) for 37 discrete and continuous distributions. The result of the density function is the height at *x* of the ordinate under the density curve of the specified distribution. Cumulative distribution functions compute the probability that a random value from the specified distribution falls below or is equal to the given value. **Inverse**

distribution functions take a specified alpha (a probability value between 0 and 1) and return the critical value x , below which lies that proportion of the specified distribution. **Random variate functions** generate pseudo-random variates from the specified distribution.

<i>Distribution</i>	<i>Cumulative</i>	<i>Density</i>	<i>Inverse</i>	<i>Random data</i>
Uniform(0,1)	UCF (x,low,hi)	UDF (x,low,hi)	UIF (a,low,hi)	URN (low,hi)
Normal (0,1)	ZCF (z,loc,sc)	ZDF (z,loc,sc)	ZIF (a,loc,sc)	ZRN (loc,sc)
t	TCF (t,df)	TDF (t,df)	TIF (a,df)	TRN (df)
F	FCF (F,df1,df2)	FDF (F,df1,df2)	FIF (a,df1,df2)	FRN (df1,df2)
Chi-Square	XCF (χ^2 ,df)	XDF (χ^2 ,df)	XIF (a,df)	XRN (df)
Gamma	GCF (x,shp,sc)	GDF (x,shp,sc)	GIF (a,shp,sc)	GRN (shp,sc)
Beta	BCF (x,shp1,shp2)	BDF (x,shp1,shp2)	BIF (a,shp1,shp2)	BRN (shp1,shp2)
Exponential (0,1)	ECF (x,loc,sc)	EDF (x,loc,sc)	EIF (a,loc,sc)	ERN (loc,sc)
Logistic (0,1)	LCF (x,loc,sc)	LDF (x,loc,sc)	LIF (a,loc,sc)	LRN (loc,sc)
Studentized	SCF (x,k,df)	SDF (x,k,df)	SIF (a,k,df)	SRN (k,df)
Weibull	WCF (x,sc,shp)	WDF (x,sc,shp)	WIF (a,sc,shp)	WRN (sc,shp)
Cauchy	CCF(x,loc,sc)	CDF(x,loc,sc)	CIF(a,loc,sc)	CRN(loc,sc)
Double exponential	DECF(x,loc,sc)	DEDF(x,loc,sc)	DEIF(a,loc,sc)	DERN(loc,sc)
Gompertz	GOCF(x,b,c)	GODF(x,b,c)	GOIF(a,b,c)	GORN(b,c)
Gumbel	GUCF(x,loc,sc)	GUDF(x,loc,sc)	GUIF(a,loc,sc)	GURN(loc,sc)
Inverse Gaussian(Wald)	IGCF(x,loc,sc)	IGDF(x,loc,sc)	IGIF(a,loc,sc)	IGRN(x,loc,sc)
Logit normal	ENCF(x,loc,sc)	ENDF (x,low,hi)	ENIF (a,loc,sc)	ENRN (loc,sc)
Lognormal	LNCF(x,loc,sc)	LNDF(x,loc,sc)	LNIF(a,loc,sc)	LNRN(loc,sc)
Pareto	PACF(x,thr,shp)	PADF(x,thr,shp)	PAIF(a,thr,shp)	PARN(thr,shp)
Rayleigh	RCF(x,sc)	RDF(x,sc)	RIF(a,sc)	RRN(sc)
Triangular	TRCF(x,a,b,c)	TRDF(x,a,b,c)	TRIF(a,a,b,c)	TRRN(a,b,c)
Loglogistic	LOCF(x,logsc,shp)	LODF(x,logsc,shp)	LOIF(a,logsc,shp)	LORN(logsc,shp)
Erlang	ERCF(x,shp,sc)	ERDF(x,shp,sc)	ERIF(a,shp,sc)	ERRN(shp,sc)
Non-central Chi-square	NXCF(x,df,d)	NXDF(x,df,d)	NXIF(a,df,d)	NXRN(df,d)
Non-central F	NFCF(x,df1,df2,d)	NFDF(x,df1,df2,d)	NFIF(a,df1,df2,d)	NFRN(df1,df2,d)
Non-central t	NTCF(x,df,d)	NTDF(x,df,d)	NTIF(a,df,d)	NTRN(df,d)
Smallest extreme value	SECF(x,loc,sc)	SEDF(x,loc,sc)	SEIF(a,loc,sc)	SERN(loc,sc)

Studentized maximum modulus	SMCF(x,k,df)	SMDF(x,k,df)	SMIF(a,k,df)	SMRN(k,df)
Binomial	NCF(x,n,p)	NDF(x,n,p)	NIF(a,n,p)	NRN(n,p)
Poisson	PCF(x,l)	PDF(x,l)	PIF(a,l)	PRN(l)
Discrete uniform	DUCF(x,N)	DUDF(x,N)	DUIF(a,N)	DURN(N)
Geometric	GECF(x,p)	GEDF(x,p)	GEIF(a,p)	GERN(p)
Hypergeometric	HCF(x,N,m,n)	HDF(x,N,m,n)	HIF(a,N,m,n)	HRN(N,m,n)
Negative binomial	NBCF(x,k,p)	NBDF(x,k,p)	NBIF(a,k,p)	NBRN(k,p)
Benford's Law	BLCF(x,B)	BLDF(x,B)	BLIF(a,B)	BLRN(B)
Logarithmic series	LSCF(x,theta)	LSDF(x,theta)	LSIF(a,theta)	LSRN(theta)
Zipf	ZICF(x,shp)	ZIDF(x,shp)	ZIIF(a,shp)	ZIRN(shp)

Here, *low* is the smallest value and *hi*, the largest value; *loc* is the location parameter and *sc*, the scale parameter; *shp* is the shape parameter and *thr*, the threshold parameter; and finally, *df* is the degrees of freedom. If *low*, *hi*, *loc*, or *sc* is omitted, the default values, which are displayed in the Distribution column, are assumed.

Expressions for the Functions

Cumulative distribution function (CDF) for discrete distributions is calculated by

$$P(X \leq x) = \sum_{y \leq x} PDF(y)$$

Inverse cumulative distribution function for discrete distributions is obtained by the smallest *x* such that

$$\sum_{y \leq x} PDF(y) \geq p$$

In expressions below, the following will hold:

- *G* denotes Gamma function.
- $p \in (0, 1)$ for calculating inverse cumulative distribution function.
- Pdf is zero when it is not specified.

Univariate Discrete Distributions

■ Benford's law

PDF:

$$f(x) = \begin{cases} \frac{\ln(1 + 1/x)}{\ln(B)} & x \in \{1, 2, 3, \dots, B-1\} \\ 0 & \text{otherwise} \end{cases}$$

Parameter(s):

B : Base $\rightarrow \{3, 4, \dots, 16\}$

CDF:

$$F(x) = \begin{cases} 0 & x < 1 \\ \frac{\ln(1 + [x])}{\ln(B)} & 1 \leq x < B-1 \\ 1 & x \geq B-1 \end{cases}$$

INVCDF:

$$F^{-1}(p) = [\exp\{p \ln(B)\}] , \text{ where } [*] \text{ is integer part } *$$

■ Binomial distribution

PDF:

$$f(x) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x} \quad x \in \{0, 1, 2, \dots, n\}$$

Parameter(s):

n : number of trials $\rightarrow \{1, 2, 3, \dots\}$

p : probability of success $\rightarrow (0, 1)$

■ Discrete uniform distribution

PDF:

$$f(x) = \frac{1}{N} \quad x \in \{1, 2, 3, \dots, N\}$$

Parameter(s):

N : number of points --- $\{1, 2, 3, \dots\}$

■ Geometric distribution

PDF:

$$f(x) = p(1-p)^x \quad x \in \{0, 1, 2, \dots\}$$

Parameter(s):

p : probability of success --- $(0, 1)$

■ Hypergeometric distribution

PDF:

$$f(x) = \frac{\frac{m!}{x!(m-x)!} \frac{(N-m)!}{(n-x)!(N-m-n+x)!}}{\frac{N!}{n!(N-n)!}} \quad \max(0, n-N+m) \leq x \leq \min(n, m)$$

Parameter(s):

N : population size --- $\{1, 2, 3, \dots\}$

m : number of units having some specific property --- $\{1, 2, 3, \dots\}$

n : sample size --- $\{1, 2, 3, \dots\}$

$$N \geq \max(m, n)$$

■ Logarithmic series distributions

PDF:

$$f(x) = \frac{a\theta^x}{x}, \quad x = 1, 2, 3, \dots, \text{ where } a = 1/n \left(\frac{1}{1-\theta} \right)$$

Parameter(s):

θ : --- $(0, 1)$

■ Negative binomial distribution

PDF:

$$f(x) = \left(\frac{\Gamma(k+x)}{\Gamma(x+1)\Gamma(k)} \right) p^k (1-p)^x \quad x \in \{0, 1, 2, \dots\}$$

Parameter(s):

k : $---(0, \infty)$

p : probability $---(0, 1)$

■ Poisson distribution

PDF:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!} \quad x \in \{0, 1, 2, \dots\}$$

Parameter(s):

λ : mean $---(0, \infty)$

■ Zipf distribution

PDF:

$$f(x) = \left[\sum_{x=1}^{\infty} x^{-(a+1)} \right]^{-1} x^{-(a+1)} \quad x \in \{1, 2, 3, \dots\}$$

Parameter(s):

a : shape $---(0, \infty)$

Univariate continuous distributions

For some continuous distributions like Normal, Gamma, t, F, etc., the CDF and Inverse CDF do not have closed form expressions. In such cases, the CDF and Inverse CDF are computed using numerical methods.

■ Beta distribution

PDF:

$$f(x) = \frac{\Gamma(\alpha + \beta)}{(\Gamma\alpha)(\Gamma\beta)} x^{\alpha-1} (1-x)^{\beta-1} \quad 0 < x < 1$$

Parameter(s):

a : shape 1 $---(0, \infty)$

b : shape 2 $---(0, \infty)$

■ Cauchy distribution

PDF:

$$f(x) = \frac{1}{\pi} \frac{\beta}{[\beta^2 + (x - \alpha)^2]} \quad -\infty < x < +\infty$$

Parameter(s):

a : location $---(-\infty, \infty)$

b : scale $---(0, \infty)$

CDF:

$$F(x) = \frac{1}{2} + \frac{1}{\pi} \tan^{-1} \left[\frac{x - \alpha}{\beta} \right]$$

Inverse CDF:

$$F^{-1}(p) = \alpha + \beta \tan[\pi(p - 0.5)]$$

■ Chi-square distribution

PDF:

$$f(x) = \frac{1}{\Gamma(n/2) 2^{(n/2)}} x^{(n/2)-1} \exp\left\{-\left(\frac{x}{2}\right)\right\} \quad x > 0$$

Parameter(s):

n : degrees of freedom $---\{1, 2, 3, \dots\}$

■ Double exponential (Laplace) distribution

PDF:

$$f(x) = (2\phi)^{-1} \exp\left\{-\left(\frac{|x-\theta|}{\phi}\right)\right\} \quad -\infty < x < +\infty$$

Parameter(s):

q : location $---(-\infty, \infty)$

f : scale $---(0, \infty)$

CDF:

$$F(x) = \begin{cases} 0.5 \exp\left\{-\left(\frac{\theta-x}{\phi}\right)\right\} & x \leq \theta \\ 1 - 0.5 \exp\left\{-\left(\frac{x-\theta}{\phi}\right)\right\} & x > \theta \end{cases}$$

Inverse CDF:

$$F^{-1}(p) = \begin{cases} \theta + \phi \ln(2p) & p \leq 0.5 \\ \theta - \phi \ln(2(1-p)) & p > 0.5 \end{cases}$$

■ Erlang distribution

PDF:

$$f(x) = \frac{1}{\Gamma \alpha \beta^\alpha} x^{\alpha-1} \exp\left\{-\left(\frac{x}{\beta}\right)\right\} \quad x > 0$$

Parameter(s):

α : shape $---\{1, 2, 3, \dots\}$

b : scale $---(0, \infty)$

■ Exponential distribution

PDF:

$$f(x) = \lambda^{-1} \exp\left\{-\left(\frac{x-\theta}{\lambda}\right)\right\} \quad x \geq \theta$$

Parameter(s):

q : location $---(-\infty, \infty)$

l : scale $---(0, \infty)$

CDF:

$$F(x) = \begin{cases} 0 & x \leq \theta \\ 1 - \exp\left\{-\left(\frac{x-\theta}{\lambda}\right)\right\} & x > \theta \end{cases}$$

Inverse CDF:

$$F^{-1}(p) = \theta - \lambda \ln(1-p)$$

■ F distribution

PDF:

$$f(x) = \frac{\left\{\Gamma\left(\frac{n_1+n_2}{2}\right)\right\} \left(\frac{n_1}{n_2}\right)^{\frac{n_1}{2}} x^{\frac{n_1}{2}-1}}{\Gamma\left(\frac{n_1}{2}\right)\Gamma\left(\frac{n_2}{2}\right) \left(1 + \frac{n_1}{n_2}x\right)^{(n_1+n_2)/2}} \quad x > 0$$

Parameter(s):

n_1 : numerator degrees of freedom $---\{1, 2, 3, \dots\}$

n_2 : denominator degrees of freedom $---\{1, 2, 3, \dots\}$

■ Gamma distribution

PDF:

$$f(x) = \frac{1}{\beta^\alpha \Gamma \alpha} x^{\alpha-1} \exp\left\{-\left(\frac{x}{\beta}\right)\right\} \quad x > 0$$

Parameter(s):

α : shape $---(0, \infty)$

b : scale $---(0, \infty)$

■ Gompertz distribution

PDF:

$$f(x) = bc^x \exp\left\{-\frac{b}{\ln c}(c^x - 1)\right\} \quad x \geq 0$$

Parameter(s):

$$b: ---(0, \infty)$$

$$c: ---(1, \infty)$$

CDF:

$$F(x) = \begin{cases} 0 & x \leq 0 \\ 1 - \exp\left\{-\frac{b}{\ln c}(c^x - 1)\right\} & x > 0 \end{cases}$$

Inverse CDF:

$$F^{-1}(p) = \frac{\ln\left(1 - \frac{\ln c \ln(1-p)}{b}\right)}{\ln c}$$

■ Gumbel distribution

PDF:

$$f(x) = \frac{1}{\theta} \exp\left\{-\left(\frac{x-\alpha}{\theta}\right)\right\} \exp\left\{-\exp\left\{-\left(\frac{x-\alpha}{\theta}\right)\right\}\right\} \quad -\infty < x < +\infty$$

Parameter(s):

$$\alpha: \text{location } ---(-\infty, \infty)$$

$$\theta: \text{scale } ---(0, \infty)$$

CDF:

$$F(x) = \exp\left\{-\exp\left\{-\left(\frac{x-\alpha}{\theta}\right)\right\}\right\}$$

Inverse CDF:

$$F^{-1}(p) = \alpha - \theta \ln(-\ln p)$$

■ **Inverse Gaussian (Wald) distribution**

PDF:

$$f(x) = \left(\frac{\lambda}{2x^3\pi} \right)^{\frac{1}{2}} \exp\left\{ -\frac{\lambda}{2\mu^2 x} (x - \mu)^2 \right\} \quad x > 0$$

Parameter(s):

m : location $---(0, \infty)$

l : scale $---(0, \infty)$

CDF:

$$F(x) = \begin{cases} 0 & x \leq 0 \\ \Phi\left\{ \sqrt{\frac{\lambda}{x}} \left(\frac{x}{\mu} - 1 \right) \right\} + \exp\left\{ 2\lambda/\mu \right\} \Phi\left\{ -\sqrt{\frac{\lambda}{x}} \left(\frac{x}{\mu} + 1 \right) \right\} & x > 0 \end{cases}$$

where $\Phi(x)$ denotes the cdf of normal(0,1) distribution.

■ **Logistic distribution**

PDF:

$$f(x) = \beta^{-1} \exp\left\{ -\left(\frac{x - \alpha}{\beta} \right) \right\} \left[1 + \exp\left\{ -\left(\frac{x - \alpha}{\beta} \right) \right\} \right]^{-2} \quad -\infty < x < +\infty$$

Parameter(s):

a : location $---(-\infty, \infty)$

b : scale $---(0, \infty)$

CDF:

$$F(x) = \left[1 + \exp\left\{ -\left(\frac{x - \alpha}{\beta} \right) \right\} \right]^{-1}$$

Inverse CDF:

$$F^{-1}(p) = \alpha - \beta \ln\left(\frac{1}{p} - 1\right)$$

■ **Logit normal distribution**

PDF:

$$f(x) = \frac{1}{\sqrt{2\pi} \sigma x (1-x)} \exp\left\{-\frac{1}{2} \left(\frac{\ln\left(\frac{x}{1-x}\right) - \mu}{\sigma}\right)^2\right\} \quad 0 < x < 1$$

Parameter(s)

m : location $---(-\infty, \infty)$

s : scale $---(0, \infty)$

■ **Loglogistic distribution**

PDF:

$$f(x) = \frac{1}{x\beta} \frac{\exp\left\{-\left(\frac{\ln x - \alpha}{\beta}\right)\right\}}{\left(1 + \exp\left\{-\left(\frac{\ln x - \alpha}{\beta}\right)\right\}\right)^2} \quad x > 0$$

Parameter(s):

a : log of scale $---(-\infty, \infty)$

b : shape $---(0, \infty)$

■ **Lognormal distribution**

PDF:

$$f(x) = \frac{1}{\sqrt{2\pi} \sigma x} \exp\left\{-\frac{1}{2} \left(\frac{\ln x - \mu}{\sigma}\right)^2\right\} \quad x > 0$$

Parameter(s):

m : location --- $(-\infty, \infty)$

s : scale --- $(0, \infty)$

■ Normal distribution

PDF:

$$f(x) = \frac{1}{\sqrt{2\pi} \sigma} \exp\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\} \quad -\infty < x < +\infty$$

Parameter(s):

m : location --- $(-\infty, \infty)$

s : scale --- $(0, \infty)$

■ Non-central chi-square distribution

PDF

$$f(x) = \frac{\exp\left\{-\frac{1}{2}(x+\delta)\right\}}{2^{\frac{n}{2}}} \sum_{j=0}^{\infty} \left(\frac{x^{\frac{n}{2}+j-1} \delta^j}{\Gamma(\frac{n}{2}+j) 2^{2j} j!} \right) \quad x > 0$$

Parameter(s):

n : degree of freedom --- $(0, \infty)$

d : non-centrality parameter --- $(0, \infty)$

■ Non-central F distribution

PDF:

$$f(x) = \sum_{j=0}^{\infty} \frac{e^{-\delta} \delta^j}{j!} \frac{\Gamma(\frac{n_1+n_2}{2}+j)}{\Gamma(n_1/2+j)\Gamma(n_2/2)} \left(\frac{n_1}{n_2}\right)^{\frac{n_1}{2}+j} \frac{x^{\frac{n_1}{2}+j-1}}{(1+\frac{n_1}{n_2}x)^{\frac{n_1+n_2}{2}+j}} \quad x > 0$$

Parameter(s):

n_1 : numerator degree of freedom--- $(0, \infty)$

n_2 : denominator degree of freedom--- $(0, \infty)$

d : non-centrality parameter--- $(0, \infty)$

■ Non-central t distribution

PDF:

$$f(x) = \frac{1}{\sqrt{n\pi}} \sum_{j=0}^{\infty} \frac{e^{-\delta^2/2} (\delta x)^j}{j!} \left(\frac{2}{n}\right)^{j/2} \frac{\Gamma(\frac{n+j+1}{2})}{\Gamma(n/2)} \frac{1}{\left(1 + \frac{x^2}{n}\right)^{\frac{n+j+1}{2}}} \quad -\infty < x < \infty$$

Parameter(s):

n : degree of freedom--- $(0, \infty)$

d : non-centrality parameter--- $(-\infty, \infty)$

■ Pareto distribution

PDF:

$$f(x) = \frac{\beta \alpha^\beta}{x^{\beta+1}} \quad x \geq \alpha$$

Parameter(s):

a : threshold --- $(0, \infty)$

b : shape --- $(0, \infty)$

CDF:

$$F(x) = \begin{cases} 0 & x \leq \alpha \\ 1 - \left(\frac{\alpha}{x}\right)^\beta & x > \alpha \end{cases}$$

Inverse CDF:

$$F^{-1}(p) = \frac{\alpha}{(1-p)^{1/\beta}}$$

■ **Rayleigh distribution**

PDF:

$$f(x) = \frac{1}{\sigma^2} x \exp\left\{-\left(\frac{x^2}{2\sigma^2}\right)\right\} \quad x \geq 0$$

Parameter(s):

σ : scale $\rightarrow (0, \infty)$

CDF:

$$F(x) = \begin{cases} 0 & x \leq 0 \\ 1 - \exp\left\{-\left(\frac{x^2}{2\sigma^2}\right)\right\} & x > 0 \end{cases}$$

Inverse CDF:

$$F^{-1}(p) = \sigma (-2 \ln(1-p))^{1/2}$$

■ **Smallest extreme value distribution**

PDF:

$$f(x) = \frac{1}{\theta} \exp\left\{\left(\frac{x-\alpha}{\theta}\right) - \exp\left\{\left(\frac{x-\alpha}{\theta}\right)\right\}\right\} \quad -\infty < x < +\infty$$

Parameter(s):

α : location $\rightarrow ((-\infty, \infty))$

θ : scale $\rightarrow (0, \infty)$

CDF:

$$F(x) = 1 - \exp\{-\exp\{\left(\frac{x-\alpha}{\theta}\right)\}\} \quad +\infty < x < +\infty$$

INVCDF:

$$F^{-1}(p) = \alpha + \theta \ln(-\ln(1-p))$$

■ **Studentized maximum modulus distribution**

PDF:

$$F(x) = \int_0^{\infty} [2\Phi(xy) - 1]^k g(y, v) dy$$

$$\text{where } \Phi(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^t \exp\{-z^2/2\} dz$$

and $g(y, v)$ is the PDF of Y , where vY^2 is chi-squared distributed with v degrees of freedom.

Parameter(s):

k : sample size --- $\{2, 3, \dots\}$

v : degrees of freedom --- $\{1, 2, \dots\}$

■ **Studentized range distribution**

CDF:

$$F(s) = C \int_0^{\infty} x^{v-1} e^{-(vx^2)/2} \left\{ k \int_{-\infty}^{\infty} \theta(y) [\Phi(y) - \Phi(y-sx)]^{k-1} dy \right\} dx \quad s > 0$$

where

$$C = \frac{V^{V/2}}{[\Gamma(V/2)][2^{V/2-1}]}$$

$$\theta(y) = \frac{1}{\sqrt{2\pi}} \exp\{-y^2/2\}$$

and

$$\Phi(y) = \int_{-\infty}^y \theta(t) dt$$

Parameter(s):

V : degrees of freedom --- $\{1, 2, 3, \dots\}$

k : number of samples --- $\{2, 3, 4, \dots\}$

■ t distribution

PDF:

$$f(x) = \frac{\Gamma(\frac{n+1}{2})}{\Gamma(n/2)\sqrt{\pi n}} \left[1 + \frac{x^2}{n}\right]^{-(n+1)/2} \quad -\infty < x < +\infty$$

Parameter(s):

n : degrees of freedom --- $\{1, 2, 3, \dots\}$

■ Triangular distribution

PDF:

$$f(x) = \begin{cases} \frac{2(x-a)}{(b-a)(c-a)} & a \leq x < c \\ \frac{2(b-x)}{(b-a)(b-c)} & c \leq x \leq b \end{cases}$$

Parameter(s):

a : minimum or low --- $(-\infty, \infty)$

b : maximum or high --- $(-\infty, \infty)$

c : mode --- $(-\infty, \infty)$

$a < c < b$

CDF:

$$F(x) = \begin{cases} 0 & x \leq a \\ \frac{(x-a)^2}{(b-a)(c-a)} & a < x < c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)} & c \leq x < b \\ 1 & x \geq b \end{cases}$$

Inverse CDF:

$$F^{-1}(p) = \begin{cases} a + (b-a) \sqrt{\frac{c-a}{b-a} p} & p \leq \frac{c-a}{b-a} \\ a + (b-a) \left(1 - \sqrt{\left(1 - \frac{c-a}{b-a}\right)(1-p)} \right) & p > \frac{c-a}{b-a} \end{cases}$$

■ Uniform distribution

PDF:

$$f(x) = \frac{1}{b-a} \quad a \leq x \leq b$$

Parameter(s):

a : minimum or low --- $(-\infty, \infty)$

b : maximum or high --- $(-\infty, \infty)$

$a < b$

CDF:

$$F(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a < x < b \\ 1 & x \geq b \end{cases}$$

Inverse CDF:

$$F^{-1}(p) = a + p(b-a)$$

■ Weibull distribution

PDF:

$$f(x) = \frac{\alpha}{\beta} \left(\frac{x}{\beta}\right)^{\alpha-1} \exp\left\{-\left(\frac{x}{\beta}\right)^{\alpha}\right\} \quad x > 0$$

Parameter(s):

α : shape $\rightarrow (0, \infty)$

β : scale $\rightarrow (0, \infty)$

CDF:

$$F(x) = \begin{cases} 0 & x \leq 0 \\ 1 - \exp\left\{-\left(\frac{x}{\beta}\right)^{\alpha}\right\} & x > 0 \end{cases}$$

Inverse CDF:

$$F^{-1}(p) = \beta (\ln(1-p))^{1/\alpha}$$

The algorithms for all functions are chosen to favor numerical accuracy over speed. Some functions may take more time to evaluate for some choices of parameters.

For more information about distributions, refer Evans et al. (2000), Johnson et al. (2005), Johnson et al. (1994), Johnson et al. (1995), and Lund and Lund (1983).

Using Distribution Functions

- Use the probability density functions (\$df) to obtain the height at x of the ordinate under the density curve of the specified distribution. For example:

To compute the geometric density for a variable say x with $p=0.75$:

```
LET geometric = GEDF(x,0.75)
```

To compute Pareto density for a variable say y with $\alpha=20.57$ and $\beta=1.75$:

```
LET pareto=PADF(y, 20.57,1.75)
```

To compute Gumbel density for a variable say z with $\text{loc}=-8.75$ and $\text{sc}=0.5$:

```
LET gumbel=GUDF(z, -8.75,0.5)
```

- Use the cumulative distribution functions (\$cf) to obtain probabilities associated with observed sample statistics. For many applications, to obtain significance levels, you must find the upper tail probabilities. For example:

```
LET p = 1 - FCF(4.14,3,7) !! from an F(3,7) distribution
```

```
LET p = 1 - FCF(4.14,3,7) !! from a t(8) distribution
```

```
LET p = 1 - ZCF(2.95,4,0.25) !! from a N(4,0.25) distribution
```

- Use the inverse distributions (\$if) to determine the critical values and to construct confidence intervals. They are also handy for power calculations.
- You can use transformations for the random variate functions to produce random deviates from the desired distributions. To produce a random normal deviate with the required mean and standard deviation, use an expression like:

```
LET normal = mean + stddev*ZRN()
```

or

```
LET norm = ZRN(10,2)
```

To produce a uniform deviate between a and b , use an expression like:

```
LET unifrmab = a + (b-a)*URN()
```

or

```
LET unifrmab = URN(a,b)
```


To get an extreme value variate with location ALPHA and scale BETA, use:

```
LET extreme = alpha - beta*LOG(ERN())
```

or

```
LET extreme = GURN(alpha ,beta)
```

Random Number Generators

SYSTAT offers two algorithms for random number generation - the Wichmann-Hill (1982) algorithm from earlier versions and the more modern Mersenne-Twister (MT). Many features such as bootstrapping, random sampling from standard distributions and Monte Carlo computations depend crucially on the efficacy of the random number generator used. SYSTAT gives users the option between these two generators, which has to be exercised as a global option.

The Wichmann-Hill algorithm generates pseudorandom numbers by a triple modulo method. Each uniform variate is constructed from three multiplicative congruential generators with prime modulus. The initial seeds for each generator are 13579, 12345 and 131. You can modify the first seed of this generator by using RSEED:

```
RSEED seed
```

where *seed* is a positive integer.

Mersenne-Twister (MT) is a pseudorandom number generator developed by Makoto Matsumoto and Takuji Nishimura (1998). It is believed to have a far longer period and a far higher order of equidistribution than any of the other implemented generators. (It has been proved that the period is $2^{19937}-1$; and 623-dimensional equidistribution property is assured.) Here too you can mention your own seed by using RSEED:

```
RSEED seed
```

where *seed* is any positive integer from 1 to $2^{32}-1 = 4294967295$.

Mersenne-Twister is the default option. We recommend the MT option, especially if the number of uniform random numbers to be generated for your Monte Carlo exercise is large, say more than 10,000.

If you would like to reproduce results involving random number generation from earlier SYSTAT versions, with old command files or otherwise, make sure that your random number generation option (under Edit=> Options=> General=>Random

Number Generation=>) is Wichmann-Hill (and, of course, that your seed is the same as before.)

For the Wichmann-Hill generator, RSEED can be any positive integer from 1 to 30000. You can change the random number generator using RNDGEN:

```
RNDGEN WH
```

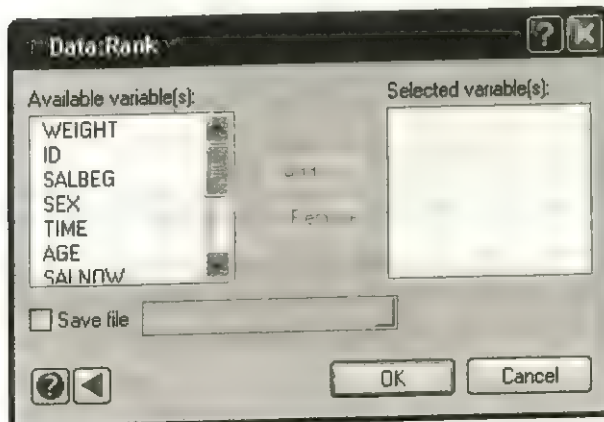
Rank Dialog Box

The rank is the case number a value has if the data are sorted by that variable. For example, the smallest value has rank 1, and the largest value in a sample of 15 cases has rank 15. Ties are averaged. This example shows how ties are handled:

<i>Value</i>	<i>Rank</i>
1	1
2	2
3	3.5
3	3.5
10	5
11	6

To open the Rank dialog box, from the menus choose:

Data
Rank...



RANK replaces the values of one or more variables with their rank values. Only the selected variables are affected; all other variables are copied unchanged to the new file.

Save file. You can save the output in a data file. Otherwise, the transformed data are stored in a temporary file.

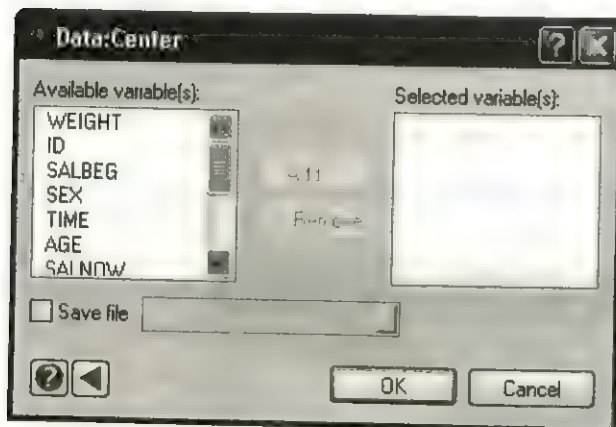
If you want to have both the ranks and the original values in one file, use LET to make a copy of the variable before ranking.

Center Dialog Box

You can center variables around their means, using Center from the Data menu (or CENTER command).

To open the Center dialog box, from the menus choose:

Data
Center...



Center subtracts the mean of the specified variable from each value of that variable.

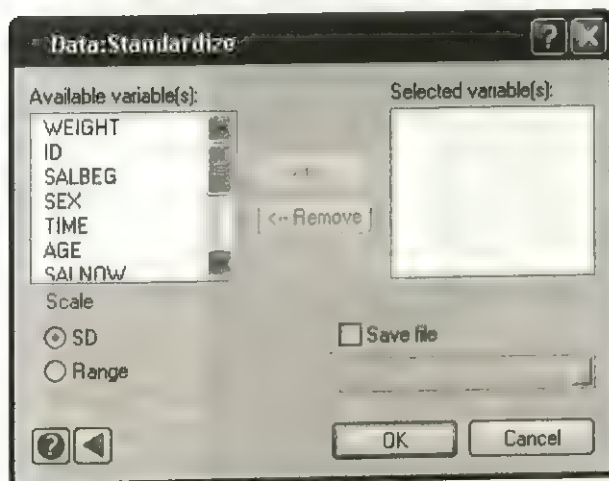
Save file. Saves the centered data in a data file. If you do not select this option, SYSTAT saves the centered data in a temporary file. If you want both centered values and values of the original variable, use LET to make a copy of the variable before centering.

Standardize Dialog Box

You can standardize one or more variables with Standardize from the Data menu (or the STANDARDIZE command).

To open the Standardize dialog box, from the menus choose:

Data
Standardize...



Standardize replaces the values of each specified variable with its sample standard score (z score) or range-standardized scores. The following options are available:

SD. For each variable, SYSTAT subtracts the variable's sample mean from each value and then divides the difference by the sample standard deviation. The standardized values have a mean of 0 and a standard deviation of 1. Standard scores are the default standardization measures.

Range. Subtracts the smallest data value of each variable from each value and divides by its range. The new scale starts at 0 and ends at 1.0.

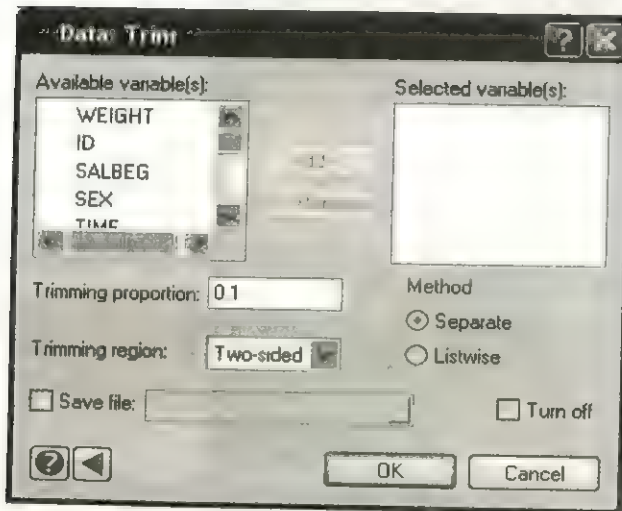
Save file. Saves the standardized data in a data file. If you do not select this option, SYSTAT saves the standardized data in a temporary file. If you want both standardized scores and values of the original variable, use LET to make a copy of the variable before standardizing.

Trimming Data

Data sets often contain a few extreme values or outliers which may influence the computed estimates or statistics of interest. In such cases, you may opt for trimming a specified proportion of extreme observations, from one side or both sides.

To open the Trim dialog box, from the menus choose:

Data
Trim ...



Trim removes (or excludes) the specified proportion (default is 0.10) of extreme observations from the data file for the selected variable(s). A two-sided 0.10 trim will trim the 0.10 proportion of the smallest and 0.10 proportion of the largest observations, not the 0.10 proportion of overall observations. You can employ one of the following two methods for trimming:

- **Separate.** Removes the specified proportion of extreme observations, separately for each of the selected variables. Here trimming of observations for one variable is independent of trimming of other variable(s). In this method, the trimmed observations are deleted and replaced by "." marks. Separate is the default trimming method.
- **Listwise.** Excludes the specified proportion of extreme observations separately for each of the selected variables at the first step, then completely excludes all those

cases which have observations for at least one of the selected variable(s) excluded. In this method, the trimmed observations are not deleted -- the cases that remain after trimming are shaded in yellow colour. For this method, you can invoke the Inverse Case Selection Toolbar (Data => Inverse Case Selection) which allows you to trim only the cases which are currently untrimmed.

Note that both the methods provide the same trimming if only one variable is trimmed.

The following options are available:

- **Trimming proportion.** Specifies the proportion of trimming. The value of the trimming proportion for two-sided trimming should lie in the range (0, 0.5). For lower or upper trimming, it should lie in the range (0,1). The default is 0.10.
- **Trimming region.** Selects whether two-sided (default) extreme observations have to be trimmed, or only the lower or upper extreme observations.
- **Save file.** Saves the trimmed data in a data file. If you do not select this option, SYSTAT saves the trimmed data in a temporary file. If you select the trimming method as "Separate", then this option saves all the cases as in the original data file with "." marks for the trimmed observations in the specified variable(s). If you select the "Listwise" method then this option saves only the shaded cases in the data editor.
- **Turn off.** Turns off case trimming so that all cases are used in subsequent analysis. You can also turn off case trimming by closing SYSTAT and opening a new data file.

Reshaping Data

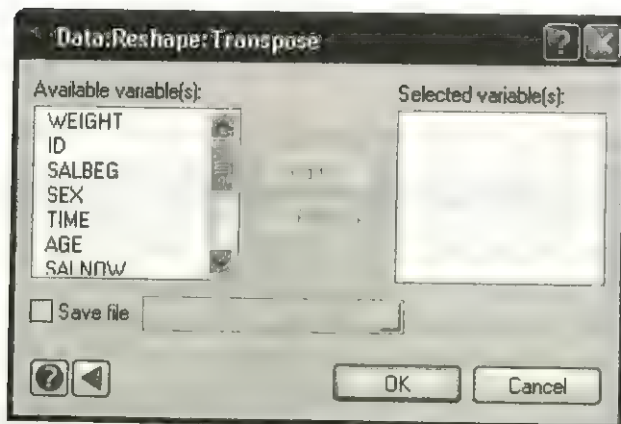
SYSTAT enables you to reshape your data file in order to make it suitable for performing the desired analysis.

Transpose

Transpose changes rows to columns and vice-versa, transposing cases and variables.

To open the Transpose dialog box, from the menus choose:

Data
Reshape
Transpose...



You can select variables to include in the transposed data file. If no variables are selected, all numeric variables are included.

You can transpose case labels into variable labels if the case labels are in a variable named *LABEL\$* in the untransposed file. The values of *LABEL\$* are used to label the variables in the transposed file. If there is no *LABEL\$* variable, the names for the columns in the new transposed file are *COL(n)* where *n* is the case number in the original (untransposed) file.

In the transposed data file, SYSTAT creates a variable (*LABEL\$*) that preserves variable names from the untransposed file. For example:

<i>X</i>	<i>Y</i>	<i>Z</i>		<i>COL(1)</i>	<i>COL(2)</i>	<i>COL(3)</i>	<i>COL(4)</i>	<i>LABEL\$</i>
10	15	4		10	6	4	8	X
6	11	9	becomes	15	11	13	9	Y
4	13	1		4	9	1	16	Z
8	9	16						

String variables (other than *LABEL\$*) are not transposed and are dropped from the resulting data file.

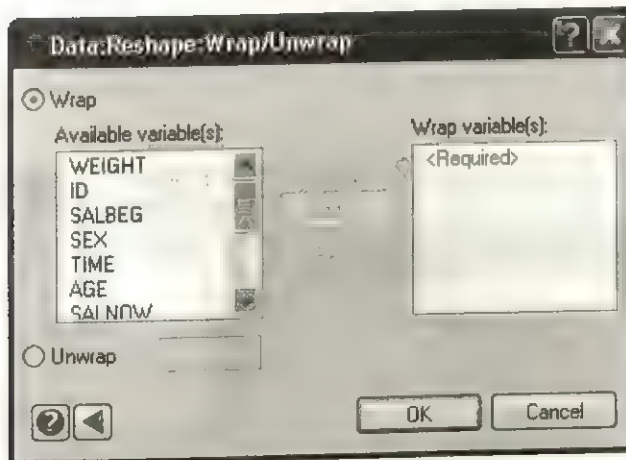
Save file. Specifies an output file for the transposed data. If you do not select this option, SYSTAT saves the data in a temporary file.

Wrap/Unwrap

Wrap/Unwrap enables you to reshape your data in order to change a multivariate repeated measures layout to a split-plot data layout (or the WRAP command) or to change a split-plot data layout to a multivariate measures layout (or the UNWRAP command).

To open the Wrap/Unwrap dialog box, from the menus choose:

Data
Reshape
Wrap/Unwrap...



Wrap. Unpacks data from each case over multiple records according to the number of variables selected.

Unwrap. Packs each block of n cases into a single record.

The left side of the following table illustrates three variables and two cases wrapped using all three variables. The right side of the table illustrates one variable with six cases unwrapped using blocks of size 3.

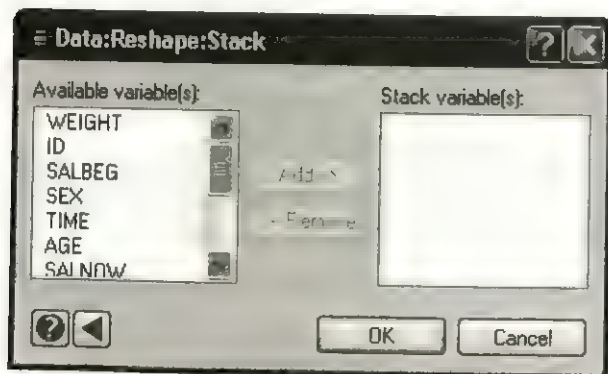
<i>WRAP</i>			<i>UNWRAP</i>		
1 2 3	Becomes	1	1	Becomes	1 2 3
4 5 6		2	2		4 5 6
		3			
		4			
		5			
		6			

Stack

Stack arranges the observations from two or more numeric variables into a single column.

To open the Stack dialog box, from the menus choose:

Data
Reshape
Stack...



You can select the variables that are to be stacked. After stacking, two columns are created: *GROUP\$*, which indicates the variable name, and *VARIABLE*, which shows the corresponding observations.

If the data file has other variables, they will be displayed after the *VARIABLE* column.

For example,

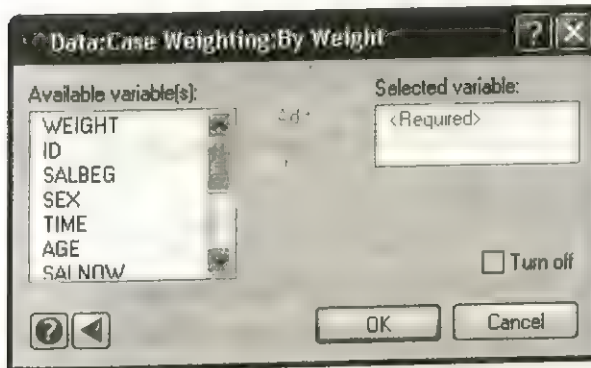
<i>X</i>	<i>Y</i>	<i>A</i>		<i>GROUP\$</i>	<i>VARIABLE</i>	<i>A</i>
8	9	15		X	8	15
3	-2	12	becomes	X	3	12
0	7	76		X	0	76
				Y	9	15
				Y	-2	12
				Y	7	76

Weight Dialog Box

Weight gives cases different weights for statistical analysis. Use Weight from the Data menu (or the WEIGHT command) to assign a weighting variable.

To open the Weight dialog box, from the menus choose:

Data
Case Weighting
By Weight...

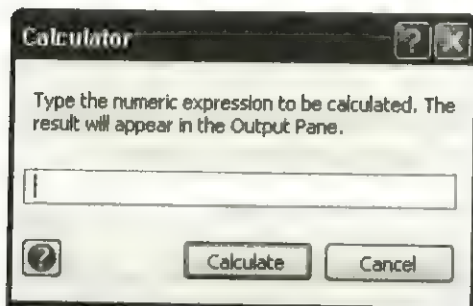


You must select a numeric value that represents the degree of importance (or weight) the case should have when performing analyses. Typically, the weight is proportional to 1 divided by the variance, when such information is known. If single cases in your data file represent multiple observations, use frequency (from the menus, Data => Case Weighting => By Frequency). Once you apply a weight variable, it remains in effect until you select another weight variable or turn weighting off. When the file type is not

rectangular, weight has no effect. If a variable is declared as a weight variable, an icon indicating the weight is displayed on the top of the variable in the Data Editor.

The Calculator

SYSTAT's calculator is available with every statistical analysis or graphical display. Use Calculator from the Utilities menu (or the CALCULATE command) to calculate.



- Balance your checkbook.
- Get the current date and time.
- Take the antilog of an output result (for example, EXP(1.826)).
- Compute your own test statistic for comparing a smaller model (S) nested within a larger model (L)—use the residual sum of squares from the two models:

$$F = \frac{(SS_S - SS_L) / (df_S - df_L)}{SS_S / df_L}$$

- Find the probability associated with the resulting F statistic.
- Try out a transformation statement to check if your formula is correct—for example, to compute percentage change:

$$\frac{\text{after} - \text{before}}{\text{before}} \times 100$$

The calculator does not know the values of any variables in your current file—it uses only the numbers you enter into it. All the functions and operators in SYSTAT are available to the calculator.

For example, to calculate today's date and time, type

```
CALC NOW$ ('mmm dd, hhhh:mm')
```

SYSTAT gives you today's date and time in the requested format.

To exponentiate the value 1.826 (that is, to compute $e^{1.826}$), type

```
CALC EXP(1.826)
```

yielding the result:

```
6.209
```

Using Commands

To transform variables, use

```
LET variable = expression
```

or

```
IF condition THEN LET assignment
```

You can specify transformation commands in any of SYSTAT's statistical or graphical procedures. Just type your transformation statements before the HOT command for that procedure. For example, to transform *GNP* per capita to log base 10 units for an ANOVA procedure

```
ANOVA
  USE OURWORLD
  LET gdp_cap = L10(gdp_cap)
  DEPEND gdp_cap
  CATEGORY gdp$
  ESTIMATE
```

The results of transformations are not automatically saved in a file. To save results using commands, type

```
DSAVE filename
```

after ESTIMATE.

To replace variable values with ranks, type

```
RANK varlist
```


To center variables, type

```
CENTER varlist
```

To standardize variables, type

```
STANDARDIZE varlist / SD or RANGE
```

To stack variables, type

```
STACK varlist
```

To trim data, type

```
TRIM varlist  
/ TRPROP =p      (0<p<1) for lower or upper and (0<p<0.5) for  
                  two-sided trimming.  
TREGION = TWOSIDED or UPPER or LOWER  
METHOD = SEPARATE or LISTWISE
```

To turn off case trimming, type

```
TRIM / OFF
```

For RANK, CENTER, STANDARDIZE, STACK, and TRIM commands, if no variable is specified, all numeric variables in the data file are used. If you want to have both the modified and the original values in one file, use LET to make a copy of the variable before issuing these commands:

```
LET age2 = age
```

To weight cases, type

```
WEIGHT var
```

To transpose cases and variables, type

```
TRANSPPOSE varlist
```

Varlist is optional and may include only *LABEL\$* and numeric variables. Notice that transposing a symmetric matrix (for example, correlations) is unnecessary, since the transpose of a symmetric matrix is the original matrix.

To wrap variables, type

```
WRAP varlist
```


To unwrap variables, into a block of n cases on a single record, type

```
UNWRAP n
```

To specify random seed, type

```
RSEED n
```

To select random number generator, type

```
RNDGEN MT or WH (Default is MT)
```

Examples

Example 1 Lagging Variables

The LAG function shifts values down n rows, replacing the first n values with a missing value:

```
LAG (var, n)
```

If n is omitted, the default is 1. Examples include:

```
LET y=LAG(x)
LET z=LOG(LAG(x))
```

The first case of a lagged variable is set to missing.

Cumulative Sum

The cumulative sum of a variable can be computed using the LAG function.

The input is:

```
NEW
REPEAT 10
LET A = CASE
IF (CASE=1) THEN LET CUSUM=A
IF (CASE>1) THEN LET CUSUM=LAG (CUSUM)+A
LIST CUSUM
```


The output is:

Case	CUSUM
1	1.000
2	3.000
3	6.000
4	10.000
5	15.000
6	21.000
7	28.000
8	36.000
9	45.000
10	55.000

Example 2

Finding the Number of Missing Values

Use the MIS function to report the number of missing values. For example, if you have survey results, use MIS to find the number of questions each respondent did not answer. Here, we use the *TESTSCOR* data file.

SUBJECTS	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	NUM_MISS
N. Smith	1	3	5	1	2	.	3	.	5	1	2
G. Henry	.	2	.	1	3	4	.	3	2	4	3
C. Bauer	3	1	4	8	9	1	5	.	2	2	1
R. McMahon	2	5	.	1	3	4	2	.	.	5	3
S. Collins	1	2	.	5	2	.	3	1	2	2	2
L. Ryan	5	4	1	2	.	.	3	.	5	1	3
E. Clark	3	2	1	2	3	3	.	4	5	1	1
T. Price	5	3	4	4	.	2	.	.	1	3	3

We create a new variable, *NUM_MISS*, containing the number of questions each respondent did not answer:

```
LET num_miss = MIS(q1 .. q10)
```

Instead of typing sequential variable names (Q1, Q2, Q3, Q4, etc.), use a double period (..) to include all variables in between (Q1 .. Q10). If you save the data, *NUM_MISS* is the last variable.

Example 3

Summary Statistics

Using four scores (S1 through S4) as data for each subject, we use the NUM, SUM, MIN, and STD functions to compute:

- **NUM_SCRS** (how many scores are present)
- **TOTAL** (the sum of their scores)
- **SMALLEST** (their minimum score)
- **SD** (the standard deviation of the scores)

The input is:

```
LET num_scrs = NUM (s1,s2,s3,s4)
LET total    = SUM (s1,s2,s3,s4)
LET smallest = MIN (s1,s2,s3,s4)
LET sd       = STD (s1,s2,s3,s4)
```

The following are the original data with results for three subjects:

NAME\$	Original Data				Derived Values			
	S1	S2	S3	S4	NUM_SCR	TOTAL	SMALLEST	SD
Smith	7	6	.	3	3	16	3	2.082
Jones	8	4	6	5	4	23	4	1.708
Wilson	2	5	3	7	4	17	2	2.217

Example 4

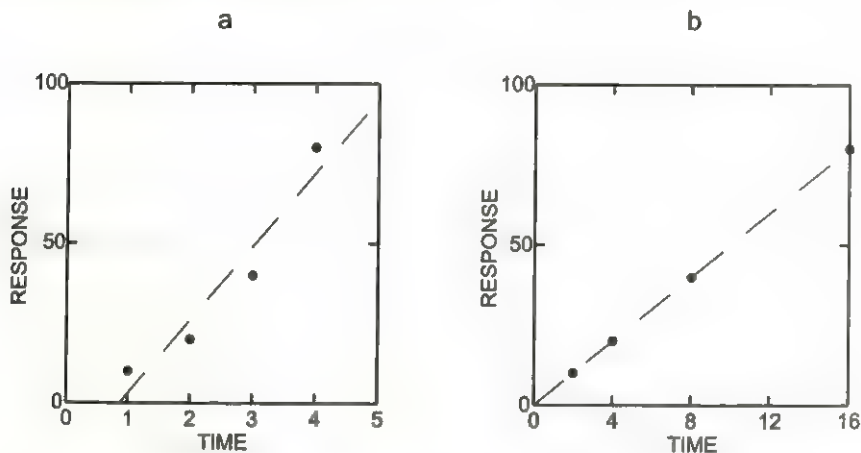
Slope of the Line of Best Fit

For each case, the SLE and SLU functions compute the slope of the line of best fit across two or more responses. Consider these four responses across four time points:

	T1	R1	T2	R2	T3	R3	T4	R4
Case 1	2	10	4	20	8	40	16	80
Case 2	2	1	4	20	8	15	16	2

The SLE function produces the slope if you assume that the times are equally spaced. To incorporate the actual spacing (2, 4, 8, 16), use the SLU function. The resulting slopes can then be used as data in statistical procedures or graphical displays—for

example, to test if the average slope for a control group differs from that for the treatment group. The following, for example, are the responses for case 1 plotted at equally spaced times and at the actual times:



To request the slope for equally spaced time points, *a*, or the slope using the actual times, *b*, specify:

```
LET slope_a = SLE(r1, r2, r3, r4)
LET slope_b = SLU(t1, r1, t2, r2, t3, r3, t4, r4)
```

You can get the same results for *SLOPE_B* by inserting the times directly:

```
LET slope_b = SLU (2, r1, 4, r2, 8, r3, 16, r4)
```

The results are:

	<i>SLOPE_A</i>	<i>SLOPE_B</i>
Case 1	23.000	5.000
Case 2	-0.200	-0.440

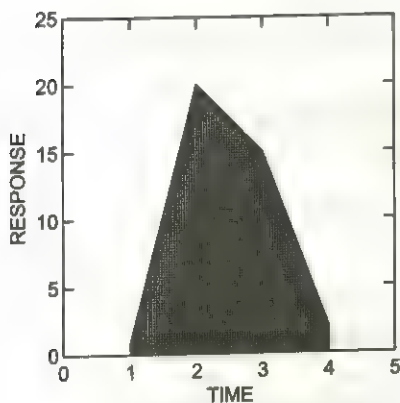
Example 5

Computing the Area under a Curve

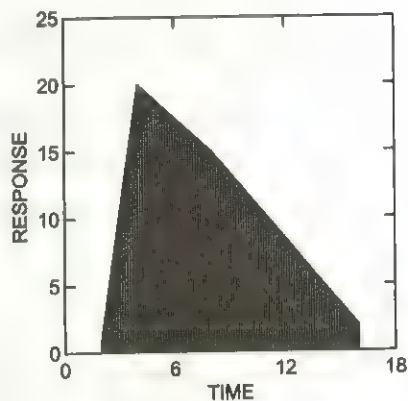
For each case, the ARE and ARU functions compute the area under the polygon formed by connecting values of variables you specify. Consider the data for Case 2:

	<i>T1</i>	<i>R1</i>	<i>T2</i>	<i>R2</i>	<i>T3</i>	<i>R3</i>	<i>T4</i>	<i>R4</i>
Case 1	2	10	4	20	8	40	16	80
Case 2	2	1	4	20	8	15	16	2

1



2



Let's use the ARE and ARU functions:

```
LET area_1 = ARE (r1, r2, r3, r4)
LET area_2 = ARU (t1, r1, t2, r2, t3, r3, t4, r4)
or
LET area_2 = ARU ( 2, r1, 4, r2, 8, r3, 16, r4)
```

The results are:

	AREA_1	AREA_2
Case 1	105.000	630.000
Case 2	36.500	159.000

Example 6

Matching Specific Values

The INC and COD functions can be used with character or numeric variables to find specified values. Within each case, the INC and COD functions search the values of selected variables for a value that matches the value you specify. When a match is found:

- INC returns a 1 (true).
- COD returns the index or order of the variable with the match (1 if the value is found in the first variable listed; 2, if it is found in the second, etc.).

If no match is found, both INC and COD return 0.

As an example, suppose respondents in a survey were asked about their favorite bath soaps:

- Is Ivory soap among the first, second, or third choices?
- What is the preference order of Ivory?

Character data. The data and results for three cases are:

NAMES	Data					Results	
	CHOICE_1\$	CHOICE_2\$	CHOICE_3\$	CHOICE_4\$	PREFRNCE\$	ORDER	PREF
Smith	Ivory	Dial	Camay	Safeguard	yes	1	1
Jones	Palmolive	Irish Spring	Camay	Ivory	no	0	0
Wilson	Zest	Ivory	Camay	Palmolive	yes	2	1

To create the new variables *PREFRNCE\$* and *ORDER*, we use the INC and COD functions:

```
LET PREFRNCE$ = 'no'
IF INC('Ivory', CHOICE_1$, CHOICE_2$, CHOICE_3$),
  THEN LET PREFRNCE$ = 'yes'
LET ORDER = COD('Ivory', CHOICE_1$, CHOICE_2$, CHOICE_3$)
```

A more direct use of the INC function is:

```
LET PREF = INC('Ivory', CHOICE_1$, CHOICE_2$, CHOICE_3$)
```

A *PREF* value of 1 indicates that Ivory is among the first three choices, and a value of 0 indicates that Ivory is not one of the first three choices.

Numeric data. For the same scenario, the data might be numeric brand codes:

1 Camay	3 Dove	5 Ivory	7 Safeguard
2 Dial	4 Irish Spring	6 Palmolive	8 Zest

The data and results are:

NAMES\$	Data				Results	
	CHOICE_1	CHOICE_2	CHOICE_3	CHOICE_4	PREFRNCS\$	ORDER
Smith	5	2	1	7	yes	1
Jones	6	4	1	5	no	0
Wilson	8	5	1	6	yes	2

The commands to obtain *PREFRNCS\$* and *ORDER* are:

```
LET PREFRNCS$ = 'no'
IF INC(5,CHOICE_1,CHOICE_2,CHOICE_3),
    THEN LET PREFRNCS$ = 'yes'
LET ORDER = COD(5,CHOICE_1,CHOICE_2,CHOICE_3)
```

Remember to use *DSAVE filename* if you want to save the new variables.

Example 7

Changing the Case of Character Values

The following examples use the *NATIONS* data file with 16 cases:

COUNTRY\$

Austria	New Guinea
Burkina Faso	New Zealand
Costa Rica	Norway
France	Portugal
Ireland	Sierra Leone
Italy	UK
Ivory Coast	USA
Netherlands	Yugoslavia

- As entered in the file.
- In capital letters.
- In lowercase letters.
- With initial capital letters.

The input is:

The output is:

Case	COUNTRY\$	COUNTRY2\$	COUNTRY3\$	COUNTRY4\$
1	Austria	AUSTRIA	austria	Austria
2	Burkina Faso	BURKINA FASO	burkina faso	Burkina faso
3	Costa Rica	COSTA RICA	costa rica	Costa rica
4	France	FRANCE	france	France
5	Ireland	IRELAND	ireland	Ireland
6	Italy	ITALY	italy	Italy
7	Ivory Coast	IVORY COAST	ivory coast	Ivory coast
8	Netherlands	NETHERLANDS	netherlands	Netherlands
9	New Guinea	NEW GUINEA	new guinea	New guinea
10	New Zealand	NEW ZEALAND	new zealand	New zealand
11	Norway	NORWAY	norway	Norway
12	Portugal	PORTUGAL	portugal	Portugal
13	Sierra Leone	SIERRA LEONE	sierra leone	Sierra leone
14	UK	UK	uk	Uk
15	USA	USA	usa	Usa
16	Yugoslavia	YUGOSLAVIA	yugoslavia	Yugoslavia

Example 8 Justifying Character Values

The following commands

```
USE NATIONS
LET country2$ = CNT$(country$,12)
LET country3$ = RGT$(country$,12)
LET country4$ = SQZ$(country3$)
LIST country$ .. country4$ / FORMAT =,
'$$$$$$$$$$$$ $$$$$$$$$$$$ $$$$$$$$$$$$ $$$$$$$$$$$$'
```

produce three new variables with values centered, right justified, and without spaces. To list these variables, we use the picture format, where \$ specifies that the values of the character variables are printed as they are stored. Note that we also used the SQZ\$ function to remove the blanks from some of the names.

The output is:

Case	COUNTRY\$	COUNTRY2\$	COUNTRY3\$	COUNTRY4\$
1	Austria	Austria	Austria	Austria
2	Burkina Faso	Burkina Faso	Burkina Faso	BurkinaFaso
3	Costa Rica	Costa Rica	Costa Rica	CostaRica
4	France	France	France	France
5	Ireland	Ireland	Ireland	Ireland
6	Italy	Italy	Italy	Italy
7	Ivory Coast	Ivory Coast	Ivory Coast	IvoryCoast
8	Netherlands	Netherlands	Netherlands	Netherlands
9	New Guinea	New Guinea	New Guinea	NewGuinea
10	New Zealand	New Zealand	New Zealand	NewZealand
11	Norway	Norway	Norway	Norway
12	Portugal	Portugal	Portugal	Portugal
13	Sierra Leone	Sierra Leone	Sierra Leone	SierraLeone
14	UK	UK	UK	UK
15	USA	USA	USA	USA
16	Yugoslavia	Yugoslavia	Yugoslavia	Yugoslavia

Example 9 Deleting Blanks or Other Characters

If you had numbers entered as character values (235,235), you could use SQZ\$ to remove the commas and then convert the character values to numeric values:

```
LET chrvar2$ = SQZ$(chrvar$, ",")
LET numvar = VAL(chrvar2$)
```

These commands convert the character value 235,235, originally stored in *CHRVAR\$*, to the number 235235 and store it in *NUMVAR*.

<code>SQZ\$(city\$)</code>	By default blanks are removed. New York becomes NewYork.
<code>SQZ\$(name\$, 'n')</code>	Remove all <i>n</i> 's.
<code>SQZ\$(name\$, ',')</code>	Remove all commas.
<code>SQZ\$(airport\$, "'")</code>	Remove all apostrophes, as in O'Hare.

Example 10

Extracting and Inserting Characters

The following commands illustrate the mechanics of the MID\$, PUT\$, RPD\$, and LPD\$ functions. These functions operate on the values of the variable *CHARDATA\$* shown on the left in the table below:

```
USE CHARDATA  
LET LAST4$ = MID$(CHARDATA$, 9, 4)  
LET INSERT_X$ = SUB$(CHARDATA$, 'efgh', 'XXXX')  
LET INSERT_Y$ = PUT$(CHARDATA$, 'YYYY', 5, 4)  
LET RTPAD_X$ = RPD$(CHARDATA$, 'X')  
LET LFTPAD_Y$ = LPD$(RGT$(CHARDATA$), 'Y')  
LIST CHARDATA$ LAST4$ INSERT_X$ INSERT_Y$ /  
FORMAT='<<<<<<<<< >>>>>>>>>> <<<<<<<<< <<<<<<<<<'
```

The results for each value of *CHARDATA\$* are:

Case	CHARDATA\$	LAST4\$	INSERT_X\$	INSERT_Y\$
1	abcdefghijkl	ijkl	abcdXXXXijkl	abcdYYYYijkl
2	abc fg hijkl	ijkl	abc fg hijkl	abc YYYYijkl
3	cdefghijkl	kl	cdXXXXijkl	cdefYYYYkl
4	abcde fgh		abcdXXXX	abcdYYYY
5	123,456.78	78	123,456.78	123,YYYY78

Remembering that values of character variables can contain up to 256 characters, we make the following observations for the first case:

- *LAST4\$* contains the last four characters (*ijkl*) of the string *abcdefghijkl*. SYSTAT starts at the ninth character (*i*) and extracts four characters.
- For *INSERT_X\$*, SYSTAT replaces the string *efgh* with *XXXX*.
- To create *INSERT_Y\$*, SYSTAT starts at the fifth character (*e*) of *abcdefghijkl* and replaces four characters with *YYYY*.
- SYSTAT adds *X*'s to the right side of the value (see Data Editor).

- ### Example 11
- #### Converting Numbers from European to American Notation

Suppose that the variable *EUR_NUMS* contains numbers in European notation.

2.734.102,56

To convert these numbers to American notation, the input is:

```
LET NO_PRD$ = SQZ$(EUR_NUM$, ',.')  
LET AM_NUM$ = SUB$(NO_PRD$, ',', '.')
```

LIST EUR_NUM\$ NO_PRD\$ AM_NUM\$ AM_NUM / FORMAT=,
'<<<<<<<<<<< <<<<<<<<<< <<<<<<<<< #####.###'

The output is:

Case	EUR_NUM\$	NO_PRD\$	AM_NUM\$	AM_NUM
1	365.452,137	365452,137	365452.137	3.655E5
2	256,4	256,4	256.4	256.400
3	8957	8957	8957	8.957E3
4	2.734.102,56	2734102,56	2734102.56	2.734E6

Example 12

Extracting First Names

Suppose your file contains a variable *FULLNAME\$* with both the first and last names of each subject:

FULLNAME\$

Scout Finch

Jane Eyre

Tom Sawyer

Billy Budd

You can extract the first name of each subject and store the result in the variable *FIRST\$*. There are two ways to create *FIRST\$*.

Using MID\$. You can use *IND* and *MID\$* together to create *FIRST\$*:

```
LET INDEX = IND(FULLNAME$, ' ')  
LET FIRST$ = MID$(FULLNAME$, 1, INDEX-1)
```

The first *LET* statement finds the position of the blank between the first and last name and stores the position in the variable *INDEX*. The second *LET* statement starts with the first character of *FULLNAME\$*, extracts *INDEX-1* characters, and stores the resulting string in the variable *FIRST\$*. So, for the first case, *Scout Finch*, *INDEX* gets the value 6, since the blank is the sixth character. Then the *MID\$* function extracts the first five (*INDEX-1*) characters (Scout) of *FULLNAME\$*, and stores the string in *FIRST\$*.

Note that you can create *FIRST\$* with only one *LET* statement by embedding the *IND* function in the *MID\$* function as follows:

```
LET FIRST$ = MID$(FULLNAME$, 1, IND(FULLNAME$, ' ')-1)
```

Using PUT\$. Alternatively, you can create *FIRST\$* by using the *PUT\$* function to replace the letters in each last name with blanks:

```
LET INDEX = IND(FULLNAME$, ' ')  
LET FIRST$ = PUT$(FULLNAME$, ' ', INDEX, 13-INDEX)
```

So, for the first case, *INDEX* gets the value 6 as before. Then the *PUT\$* function starts with the sixth character and replaces the next seven (*13-INDEX*) characters with blanks. The result, *Scout*, is stored in the variable *FIRST\$*.


```
LET FIRST$ = PUT$(FULLNAME$, ' ', IND(FULLNAME$, ' '),  
13-IND(FULLNAME$, ' '))
```

```
USE MYDATA  
LET AGE$ = CAT$('_',AGE$)  
LET SEX_AGE$ = CAT$(SEX$,AGE$)  
LET SEX_AGE$ = SUB$(SEX_AGE$,'_', ' ' )  
SORT SEX_AGE$  
LIST SEX$ AGE$ SEX_AGE$ / FORMAT='  
<<<<< <<<<< <<<<<<<<<<<'
```


The output is:

Case	SEX\$	AGE\$	SEX_AGE\$
1	Female	_Adult	Female Adult
2	Female	_Adult	Female Adult
3	Female	_Child	Female Child
4	Female	_Child	Female Child
5	Female	_Teen	Female Teen
6	Male	_Adult	Male Adult
7	Male	_Child	Male Child
8	Male	_Child	Male Child
9	Male	_Teen	Male Teen

The results are as expected. To save the data, type `DSAVE filename`. Now we can use `SEX_AGE$` as a grouping variable and request plots, statistics, or other statistical analyses separately for the six groups.

Example 14

Using Soundex to Code Names

Let's say you want to check for duplicate records in a large file with format as in the file named *SOUNDEX* listed below, but you wonder about alternative spellings. First create a variable containing the Soundex code for the *LASTNAM\$* variable:

```
LET CODE$ = SND$(lastnam$)
```

Then sort the file by this new variable:

	<i>LASTNAM\$</i>	<i>INCOME</i>	<i>AGE</i>	<i>SEX\$</i>	<i>CODE\$</i>
CASE 1	Barrett	376.300	22.000	F	B630
CASE 2	Barrott	872.100	34.000	F	B630
CASE 3	Howell	987.200	36.000	F	H400
CASE 4	MacCarthy	765.100	45.000	M	M263
CASE 5	MacLoud	987.300	40.000	F	M243
CASE 6	McCarthy	765.000	24.000	M	M263
CASE 7	McCarty	367.100	38.000	F	M263
CASE 8	McLoud	987.300	40.000	F	M243
CASE 9	McLeod	542.300	32.000	M	M243
CASE 10	Stephens	683.200	30.000	F	S315
CASE 11	Stevens	743.200	55.000	M	S315
CASE 12	Wilkenson	999.300	48.000	F	W425
CASE 13	Wilkinson	235.100	48.000	M	W425

By viewing the file, you could then easily pick out duplicate records. If you are looking for a specific duplicate record, instead of creating the *CODE\$* variable, you can select all records with alternative spellings of the same name:

```
SELECT SND$('stephens') = SND$(LASTNAM$)
```

Since the Soundex code for both names (Stephens and Stevens) is S315, both cases are selected. Now, we just list the selected cases to compare *AGE*, *SEX*, and *INCOME* to be sure that we have unique records, or a duplicate spelling of the same record:

Case	LASTNAM\$	INCOME	AGE	SEX\$	CODE\$
10	STEPHENS	683.200	30.000	F	S315
11	STEVENS	743.200	55.000	M	S315

You will discover that the records for Stephens and Stevens are unique.

Soundex uses the first letter of the word to begin the code, so words such as Carson and Karson have different codes—C625 and K625. Also, Soundex is not useful for languages other than English.

Example 15 ***Generating a New File of Random Data***

You can generate a new file with random numbers. First specify *NEW* and then use *REPEAT* to specify the number of cases you want:

```
NEW
RSEED 359
REPEAT 200
LET A=ZRN()
DSAVE filename
```


Example 16

Standardizing Age

This example uses the *CHILDREN* data set:

<i>SEX\$</i>	<i>AGE</i>	<i>N</i>
Female	5	1
Female	6	4
Female	5	5
Female	3	8
Female	5	10
Female	5	13
Female	6	14
Male	6	2
Male	4	3
Male	6	6
Male	8	7
Male	6	9
Male	4	11
Male	5	12

We create and list a variable *AGESTAND* that contains the standardized values of *AGE*.

The input is:

```
USE CHILDREN
SORT SEX$
LET AGESTAND=AGE
STANDARDIZE AGESTAND
LIST AGE AGESTAND
```


The output is:

Case	AGE	AGESTAND
1	5.000	-0.237
2	6.000	0.593
3	5.000	-0.237
4	3.000	-1.898
5	5.000	-0.237
6	5.000	-0.237
7	6.000	0.593
8	6.000	0.593
9	4.000	-1.068
10	6.000	0.593
11	8.000	2.254
12	6.000	0.593
13	4.000	-1.068
14	5.000	-0.237

AGESTAND now has standardized age values with a mean of 0 and a standard deviation of 1. Remember that standardizing does not change the shape of the distribution. If the data are highly skewed or bimodal before standardizing, they will be so after. Standardizing simply moves the location and rescales the spread of your values.

Example 17 Computing *F* Statistic

In other examples, we use the *LONGLEY* data to fit a multiple regression model with six independent variables and use forward stepping to select a subset of three predictors. The residual sum of squares for the six-variable model is 836,424.056 with 9 *df*; for the three-variable model, the value is 1,323,360.743 with 12 *df*. Let's construct an *F* statistic to compare the two models (that is, to test $b_4 = b_5 = b_6 = 0$):

$$\text{CALC } ((1323360.743 - 836424.056)/3) / (836424.056/9)$$

The calculator returns 1.746.

Finding the *p*-value associated with *F*. This expression requests the *p*-value for $F = 1.746$ with numerator *df* = 3 and denominator *df* = 9:

$$\text{CALC } 1 - \text{FCF}(1.746, 3, 9)$$

The calculator returns the probability 0.227. The inclusion of the extra three variables does not significantly improve the fit of the model.

Example 18**Trimming Cases for Selected Variable(s)**

To trim the 0.20 proportion of the largest cases for the variables *X* and *Y* by the Separate method in the *RAINFALL* data file, the input is:

```
USE RAINFALL
TRIM X Y / TRPROP= 0.20 TREGION = UPPER METHOD = SEPARATE
```

The result is:

<i>Case Number</i>	<i>X</i>	<i>Y</i>
1	23.900	41.000
2	43.300	52.000
3	36.300	18.700
4	40.600	.
5	57.000	40.000
6	52.500	29.200
7	46.100	51.000
8	.	17.600
9	.	46.600
10	23.700	.

Here, trimming by Separate method trimmed the 0.20 proportion of the largest observations from each of the *X* and *Y* variables separately. The largest 0.20 proportion of *X*, i.e. the 2 largest observations, 142 and 112.6, get trimmed, and the largest 0.20 proportion of *Y*, i.e. the 2 largest observations, 55 and 57, get trimmed.

To trim the same data by the Listwise method, the input is:

```
USE RAINFALL
TRIM X Y / TRPROP= 0.20 TREGION = UPPER METHOD = LISTWISE
```


The result is:

<i>Case Number</i>	<i>X</i>	<i>Y</i>
1	23.900	41.000
2	43.300	52.000
3	36.300	18.700
4	40.600	55.000
5	57.000	40.000
6	52.500	29.200
7	46.100	51.000
8	142.000	17.600
9	112.600	46.600
10	23.700	57.000

Here, trimming by the Listwise method first trimmed the 0.20 proportion of the largest observations from each of the *X* and *Y* variables separately and then excluded cases listwise i.e. completely excluded all those cases, which had observations excluded for at least one of *X* or *Y* previously. Thus, cases 8 and 9 which are trimmed by the Separate method for *X* only, are also excluded for *Y* and, cases 4 and 10 which are trimmed by the Separate method for *Y* only are also excluded for *X*. Only the shaded cases will be used in further analysis.

Example 19 **Stacking Variables**

To conduct a 2-sample t-test on the *WORLDDM* data set, to see if the years of life expectancy for males (*MALE*) and the years of life expectancy for females (*FEMALE*) have the same mean, the two variables have to be stacked in one column. For this the input is:

```
USE WORLDDM
STACK MALE FEMALE
```


The result is:

	GROUPS	VARIABLE	COUNTRIES	BIRTH_RT	DEATH_RT	GOVS	URBANS	LAT	LON
MALE	71	Finland		13	10	Democracy	city	-32	-64
MALE	73	France		14	9	Democracy	city	-23	133
MALE	75	Spain		11	8	Democracy	city	13	-59
MALE	73	UK		14	11	Democracy	city	50	4
MALE	74	Italy		10	9	Democracy	city	9	2
MALE	75	Sweden		13	11	Democracy	city	-17	-66
MALE	67	Hungary		12	13	OneParty	city	6	17
MALE	72	Germany		11	11	OneParty	city	-33	-71
MALE	46	Gambia		48	18	Democracy	rural	-4	14
MALE	66	Iraq		46	7	OneParty	city	9	-84
MALE	49	Ethiopia		45	15	Military	rural	49	16
MALE	40	Guinea		47	22	Military	rural	56	10
MALE	45	Mali		51	21	OneParty	rural	0	-78
MALE	65	Libya		37	7	Military	city	13	-89
MALE	53	Somalia		47	15	OneParty	rural	9	38
MALE	51	Sudan		44	14	Democracy	rural	66	25
MALE	64	Turkey		29	8	Military	city	48	2
MALE	61	Algeria		37	9	OneParty	city	0	12
MALE	48	Yemen		52	17	Military	rural	13	-15
MALE	67	Argentina		20	9	Democracy	city	9	0
MALE	73	Barbados		18	8	OneParty	city	40	22
MALE	52	Bolivia		35	13	Democracy	city	14	-90
MALE	62	Brazil		26	7	Democracy	city	10	-10
MALE	74	Canada		14	7	Democracy	city	18	-72
MALE	70	Chile		21	6	Military	city	14	-87
MALE	74	CostaRica		28	4	Democracy	city	65	-18
MALE	64	Ecuador		30	7	Democracy	city	7	-5
MALE	75	Jamaica		21	5	OneParty	city	-1	36
MALE	52	Haiti		45	16	Military	rural	6	-9
MALE	69	Trinidad		28	6	Democracy	city	-18	47
FEMALE	80	Finland		13	10	Democracy	city	-32	-64
FEMALE	82	France		14	9	Democracy	city	-23	133
FEMALE	82	Spain		11	8	Democracy	city	13	-59
FEMALE	79	UK		14	11	Democracy	city	50	4
FEMALE	81	Italy		10	9	Democracy	city	9	2
FEMALE	81	Sweden		13	11	Democracy	city	-17	-66
FEMALE	75	Hungary		12	13	OneParty	city	6	17
FEMALE	79	Germany		11	11	OneParty	city	-33	-71
FEMALE	50	Gambia		48	18	Democracy	rural	-4	14
FEMALE	68	Iraq		46	7	OneParty	city	9	-84
FEMALE	52	Ethiopia		45	15	Military	rural	49	16
FEMALE	44	Guinea		47	22	Military	rural	56	10
FEMALE	47	Mali		51	21	OneParty	rural	0	-78
FEMALE	70	Libya		37	7	Military	city	13	-89
FEMALE	54	Somalia		47	15	OneParty	rural	9	38
FEMALE	55	Sudan		44	14	Democracy	rural	66	25

FEMALE	67	Turkey	29	8	Military	city	48	2
FEMALE	64	Algeria	37	9	OneParty	city	0	12
FEMALE	49	Yemen	52	17	Military	rural	13	-15
FEMALE	74	Argentina	20	9	Democracy	city	9	0
FEMALE	77	Barbados	18	8	OneParty	city	40	22
FEMALE	56	Bolivia	35	13	Democracy	city	14	-90
FEMALE	68	Brazil	26	7	Democracy	city	10	-10
FEMALE	81	Canada	14	7	Democracy	city	18	-72
FEMALE	77	Chile	21	6	Military	city	14	-87
FEMALE	79	CostaRica	28	4	Democracy	city	65	-18
FEMALE	68	Ecuador	30	7	Democracy	city	7	-5
FEMALE	79	Jamaica	21	5	OneParty	city	-1	36
FEMALE	55	Haiti	45	16	Military	rural	6	-9
FEMALE	74	Trinidad	28	6	Democracy	city	-18	47

Now, the 2-sample t-test can be run on the rearranged dataset above.

References

- Evans, M., Hastings, N., and Peacock, B. (2000). *Statistical distributions*. 3rd ed. New York: John Wiley & Sons.
- Johnson, N.L., Kemp, A.W., and Kotz, S. (2005). *Univariate discrete distributions*. 3rd ed. New York: John Wiley & Sons.
- Johnson, N.L., Kotz, S., and Balakrishnan, N. (1994). *Univariate continuous distributions*. Volume 1, 2nd ed. New York: John Wiley & Sons.
- Johnson, N.L., Kotz, S., and Balakrishnan, N. (1995). *Univariate continuous distributions*. Volume 2, 2nd ed. New York: John Wiley & Sons.
- Lund, R. E. and Lund, J. R. (1983). Probabilities and upper quantiles for the studentized range, *Applied Statistics* 32, 204-210, Algorithm AS 190.
- Matsumoto, M. and Nishimura, T. (1998). Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, *ACM Transactions on Modeling and Computer Simulation*, 8, 3-30.
- Wichmann, B.A. and Hill, I.D. (1982). An efficient and portable pseudo-random number generator. *Applied Statistics*, 31, 188-190 (Corrections, 1984, *ibid*, 33, 123).

List, Sort, and Select

Laszlo Engelman

This chapter describes these items on the Data menu:

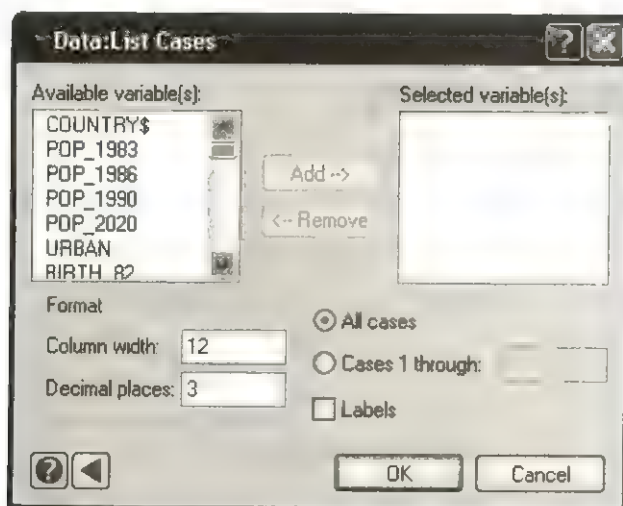
- | | |
|--------------|--|
| List Cases | Lists all or selected variables in a data file. |
| Sort File | Sorts the cases in a file according to values of one or more numeric or string variables. |
| Select Cases | Specifies a condition to select a subset of cases for analysis. For example, using <i>AGE</i> > 21 restricts subsequent analyses to adults, omitting any case where the value of <i>AGE</i> is 21 or less. |

List Cases Dialog Box

Often during the steps of an analysis, you need to stop to view data values. For example, you may want to see values of specific variables side by side for a few selected cases, check that transformations or recoding worked as intended, or scan other data values for cases with extreme residuals.

To open the List Cases dialog box, from the menus choose:

Data
List Cases...



List Cases displays the values of selected variables in the order they are selected. You can display all cases, or the first n cases up to a number you specify. If no variables are selected, all variables are listed. You can also use an ID variable to select a numeric or string variable whose values are listed before other values for each case. When you want to list more than five variables across the panel, specify a wide page format. This enables you to print up to eight variables (plus a case number or label) on each line, when Tabular Output is active.

The following additional options are available:

- **Format.** Specifies a column width to use for all variables. Any variable value shorter than this width is right-justified. Any value longer than the width is truncated. In addition, define the number of decimal places to print for numeric variables.
- **Labels.** Displays category labels instead of values.

Picture Format

To display your data, you can specify a picture format using the **FORMAT** option on the **LIST** command. You use symbols to tell SYSTAT exactly how to format your listing. This gives you a greater control over the appearance of your output. The picture format also enables you to:

- Specify the number of characters or digits in each field.

- Specify the number of spaces between variables.
- Specify the number of digits of numeric data to display after the decimal point for each variable individually.
- Choose whether to left- or right-justify the values within a field.
- Insert characters, such as vertical bars (| |), to separate columns.

You must enclose your symbol strings in quotation marks. Use a #, >, <, or \$ for each digit or character. Numeric variables are always right-justified. For string variables, the symbol you use determines whether it is left- or right-justified:

Symbol		Symbol	
blank	Separate fields	Y	Years
>	Right-justify	M	Months
<	Left-justify	D	Days
\$	Do not justify	h	Hours
.	Decimal point	m	Minutes
#	Digit	s	Seconds

If you include any other symbol (such as a vertical bar) in the FORMAT statement, it is printed “as is” for every case. Use the date symbols (Y, M, D, h, m, and s) to format dates in your output.

Additional Features for Listing Cases

When listing data, there are other options you may want to use. For example, you can:

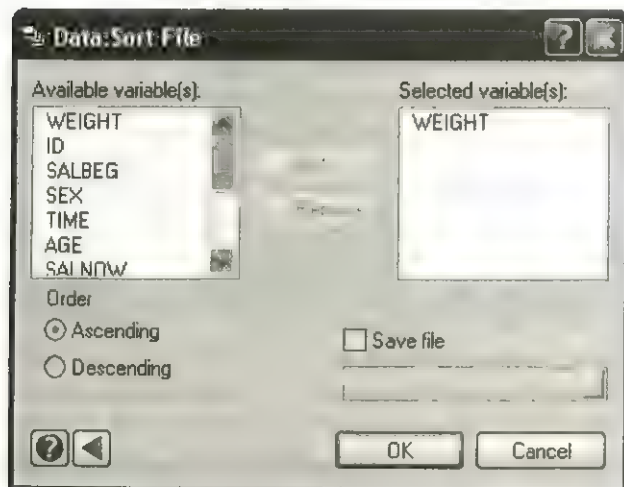
- Display eight variables per panel instead of five (the default) by specifying a wide output format (PAGE WIDE).

You can also use the PRINT command to list data values without case numbers and variable names.

Sort Dialog Box

To open the Sort dialog box, from the menus choose:

Data
Sort...



Sort orders all the cases in a file according to the variable you select. For example, you could sort cases by gender, and subsequently by age within each gender. In addition, you can specify whether values are sorted in ascending or descending order. If you do not specify any variables, SYSTAT sorts by the first variable in the file.

- **Ascending/Descending.** Sorts cases in either ascending order (1, 2, 3... or a, b, c...) or descending order (3, 2, 1...).
- **Save file.** Specifies an output file for the sorted data. If this option is selected and a filename is not specified, SYSTAT prompts you to specify a filename. If you do not select this option, SYSTAT saves the sorted data in a temporary file.

Sort Order

By default, SYSTAT orders cases in an ascending order, $(1, 2, \dots, n)$ for numeric variables and, as follows, for string variables:

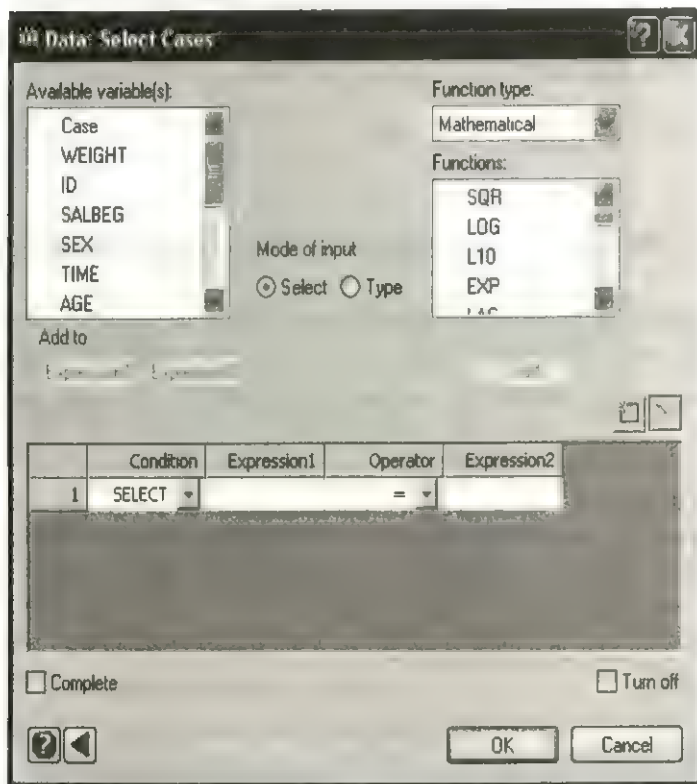
```
!"#$%&'()*+,-./
0123456789
:;<=>?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ
[\]^_
abcdefghijklmnopqrstuvwxyz
{|}~
```

- Words are sorted alphabetically with words in upper case preceding those in lower case.
- If you sort a string variable containing numeric values, those values are sorted from left to right, rather than small to big: 1, 12, 150, 2, 31, 4000, 5.4, etc.).
- Cases with missing data for the sort variable are placed at the beginning of the file.

Select Cases Dialog Box

To open the Select Cases dialog box, from the menus choose:

Data
Select Cases...



Select Cases restricts subsequent analyses to cases that meet the conditions you specify. Unselected cases remain in the data file, but are excluded from subsequent analyses until Select is turned off. For example, you could restrict your analysis to respondents of a certain age, gender, or both.

Rules for expressions. You can use any mathematically valid combination of variables, numbers, functions, and operators. You can also use any combination of selecting, pasting, and typing necessary to build the test condition. Finally, you can specify any

number of conditions, connecting them with a logical AND or OR. Use parentheses if needed for logic or clarity.

- If the expression contains any character values, they must be enclosed in single or double quotation marks. Character values are case sensitive.
- Arguments for functions must be inside parentheses, for example, LOG(weight) and SQR(income).

The following options are available:

- **Mode of Input.** Gives an option to specify selection condition by selecting available variables (in the expression) and operators or by typing the selection condition.
- **Complete.** Selects cases with no values missing.
- **Turn off.** Turns off case selection so that all cases are used in the subsequent analyses. You can also turn off case selection by closing SYSTAT, opening a new data file, or typing SELECT in the command area.

To save a new data file containing only the selected cases, from the menus choose:

Data
Extract Selection

You can also select cases in graphs using the region and lasso tools available in the selection tool of the Graph Editor. Selection can be toggled using the invert case selection icon in the data toolbar.

Using Commands

To list cases:

```
LIST varlist / FORMAT=m,n or 'picture format' N=n LABEL
```

To sort cases:

```
SORT varlist / D or A or a list of D's and A's
```

To select cases:

```
SELECT exprn1 AND exprn2 OR exprn3...
or
SELECT COMPLETE
or
SELECT CASE=expression
```


Examples

Example 1

Listing All Variables and Cases

To list all values in the *MINIWRLD* data file, the input is:

```
USE MINIWRLD
LIST
```

The first three cases of *MINIWRLD* are shown below:

Case	COUNTRY\$ EDUC LITERACY	POP_1990 HEALTH GROUP\$	URBAN MIL B_TO_D	BIRTH_RT GOV\$	BABYMORT LIFEEXPM	GDP_CAP LIFEEXPF
1	France 648.069 99.000	56.358 728.249 Europe	73.000 432.780 1.556	14.000 Democracy	6.000 73.000	14542.657 82.000
2	Greece 115.000 95.000	10.028 158.700 Europe	65.000 260.400 1.222	11.000 Democracy	10.000 75.000	5614.184 80.000
3	Switzerland 853.538 99.000	6.742 1209.077 Europe	58.000 330.769 1.333	12.000 Democracy	5.000 75.000	17723.499 83.000

The individual cases are too long to fit on one line (80 characters), so each case spreads across three lines. You can specify a wider output format using *PAGE WIDE*.

You can also display separate panels for selected variables:

```
LIST COUNTRY$ .. BABYMORT
LIST GDP_CAP .. GOV$
```

Example 2

Listing Cases Subsets for Selected Variables

Here we list only cases 1 through 10 of the variables *COUNTRY\$*, *LITERACY*, *URBAN*, and *BIRTH_RT* in the *MINIWRLD* data file.

The input is:

```
USE MINIWRLD
LIST COUNTRY$ LITERACY URBAN BIRTH_RT / N=10
```


The output is:

Case	COUNTRY\$	LITERACY	URBAN	BIRTH_RT
1	France	99.000	73.000	14.000
2	Greece	95.000	65.000	11.000
3	Switzerland	99.000	58.000	12.000
4	Spain	97.000	91.000	11.000
5	UK	99.000	76.000	14.000
6	Hungary	99.000	54.000	12.000
7	Iraq	55.000	68.000	46.000
8	Pakistan	26.000	28.000	43.000
9	Ethiopia	55.200	14.000	45.000
10	Afghanistan	12.000	16.000	44.000

Example 3

ID Variables

In this example using the *MINIWRLD* data file, we use an ID variable, *COUNTRY\$*, to identify the rows of the listing. We also use *FORMAT* to specify one decimal place in each field.

The input is:

```
USE MINIWRLD
IDVAR COUNTRY$
LIST LITERACY URBAN BIRTH_RT / N=10 FORMAT = 1
```

The output is:

Case ID	LITERACY	URBAN	BIRTH_RT
France	99.000	73.000	14.000
Greece	95.000	65.000	11.000
Switzerland	99.000	58.000	12.000
Spain	97.000	91.000	11.000
UK	99.000	76.000	14.000
Hungary	99.000	54.000	12.000
Iraq	55.000	68.000	46.000
Pakistan	26.000	28.000	43.000
Ethiopia	55.200	14.000	45.000
Afghanistan	12.000	16.000	44.000

Example 4

Group Labels

In this example using the *SURVEY2* data file, we add group labels to a listing.

The input is:

```
USE SURVEY2
LABEL MARITAL / 1="NEVER", 2="MARRIED", 3="DIVORCED", 4="SEPARATED"
LABEL EDUCATN / 1, 2="DROPOUT", 3="HS,
                GRAD", 4, 5="COLLEGE", 6, 7="DEGREE"
LIST MARITAL EDUCATN AGE SEX / N=12 LABEL
```

The output is:

Case	MARITAL	EDUCATN	AGE	SEX
1	Divorced	College	58	1
2	Married	HS grad	45	2
3	Divorced	HS grad	50	2
4	Separated	HS grad	33	2
5	Married	HS grad	24	1
6	Married	Dropout	58	2
7	Never	HS grad	22	1
8	Married	Dropout	30	1
9	Married	HS grad	57	2
10	Married	Dropout	39	1
11	Married	HS grad	23	2
12	Separated	Dropout	55	2

Example 5

Picture Format: Displaying Many Variables in One Panel

To illustrate the picture format, let us use an example from a 1980 survey of depression. In Los Angeles, interviewers from the Institute for Social Science Research at UCLA surveyed a multiethnic sample of community members for an epidemiological study of depression and help-seeking behavior among adults. The CESD depression index, used to measure depression, was constructed by asking people to respond to 20 items: "I felt I could not shake off the blues...", "My sleep was restless," etc. For each item, respondents answered "less than 1 time per day" (score 0), "1 to 2 days per week" (score 1), "3 to 4 days per week" (score 2), or "5 to 7 days" (score 3). Responses to the 20 items were summed to form a total score. Persons with a CESD total greater than or equal to 16 were classified as depressed. The information available for each subject includes the following:

Variable	Definition
<i>ID</i>	Subject identification number
<i>SEX</i>	1 = male; 2 = female
<i>AGE</i>	Age in years at last birthday
<i>MARITAL</i>	1 = never married; 2 = married; 3 = divorced; 4 = separated; 5 = widowed
<i>EDUCATN</i>	1 = less than high school; 2 = some high school; 3 = finished high school; 4 = some college; 5 = finished bachelor's degree; 6 = finished master's degree; 7 = finished doctorate
<i>EMPLOY</i>	1 = full time; 2 = part time; 3 = unemployed; 4 = retired; 5 = houseperson; 6 = in school; 7 = other
<i>INCOME</i>	Thousands of dollars per year
<i>RELIGION</i>	1 = Protestant; 2 = Catholic; 3 = Jewish; 4 = none; 5 = other
<i>BLUE to DISLIKE</i>	The 20 depression items
<i>TOTAL</i>	Total CESD score
<i>CASE</i>	0 = normal; 1 = depressed (CESD \geq 16)
<i>DRINK</i>	1 = yes, regularly; 2 = no
<i>HEALTHY</i>	General health? 1 = excellent; 2 = good; 3 = fair; 4 = poor
<i>CHRONIC</i>	Any chronic illnesses in last year? 0 = no; 1 = yes

Often it is beneficial to see many variables in a single panel so that cases can be compared. To do this, use the **FORMAT** option to specify a picture format.

The input is:

```
USE SURVEY
SORT ID
LIST ID .. TOTAL CASE DRINK / FORMAT='### # # # # # # # # | # #,
# # # # # # # # # # # # # # # # # #,
|| ## || ### #'
```


[illegible]

Example 6**Picture Format: Displaying Dates**

With the **FORMAT** option of **LIST**, you can list values in a specified date or time format. For this example, we use the **SICKDATE** data file, which lists the date each patient's illness was diagnosed (**DIAGDATE**) and the date each died (**MORTDATE**). These dates are listed in day-of-the-century format.

NAMES	DIAGDATE	MORTDATE
Jones	32153	33151
Smith	31255	32351
Williams	30251	32512
Jackson	29351	30251
Moore	28351	29351
Iverson	29351	30251
Brown	27351	32512
Long	26351	28351
Nelson	28351	30251
Dennison	24351	25351

To demonstrate how the **FORMAT** option displays these dates, we use three different formats:

MM/DD/YY MMM.DD,YYYY DD.MMM,YY

The input is:

```
USE SICKDATE
LIST DIAGDATE DIAGDATE DIAGDATE MORTDATE MORTDATE MORTDATE / FORMAT =,
'MM/DD/YYYY    MMM.DD,YYYY    DD.MMM,YY MM/DD/YYYY    MMM.DD,YYYY    DD.MMM,YY'
```

The output is:

Case	DIAGDATE	DIAGDATE	DIAGDATE	MORTDATE	MORTDATE	MORTDATE
1	01/11/2018	Jan.11,2018	11.Jan,18	10/05/2020	Oct.05,2020	05.Oct,20
2	07/28/2015	Jul.28,2015	28.Jul,15	07/28/2018	Jul.28,2018	28.Jul,18
3	10/27/2012	Oct.27,2012	27.Oct,12	01/05/2019	Jan.05,2019	05.Jan,19
4	05/11/2010	May.11,2010	11.May,10	10/27/2012	Oct.27,2012	27.Oct,12
5	08/15/2007	Aug.15,2007	15.Aug,07	05/11/2010	May.11,2010	11.May,10
6	05/11/2010	May.11,2010	11.May,10	10/27/2012	Oct.27,2012	27.Oct,12
7	11/18/2004	Nov.18,2004	18.Nov,04	01/05/2019	Jan.05,2019	05.Jan,19
8	02/22/2002	Feb.22,2002	22.Feb,02	10/27/2012	Oct.27,2012	27.Oct,12
9	08/15/2007	Aug.15,2007	15.Aug,07	10/27/2012	Oct.27,2012	27.Oct,12
10	09/01/1996	Sep.01,1996	01.Sep,96	05/29/1999	May.29,1999	29.May,99

Example 7

Simple Sort

To sort the children on *SEX\$* using the *CHILDREN* file, the input is:

```
USE CHILDREN
SORT SEX$
LIST
```

The output is:

Case	SEX\$	AGE	N
1	Female	5	1
2	Female	6	4
3	Female	5	5
4	Female	3	8
5	Female	5	10
6	Female	5	13
7	Female	6	14
8	Male	6	2
9	Male	4	3
10	Male	6	6
11	Male	8	7
12	Male	6	9
13	Male	4	11
14	Male	5	12

Cases where *SEX\$* equals *Female* come before those where *SEX\$* equals *Male*. (*N* represents the original position of each case.)

Example 8

Nested Sort

If you select more than one sort variable, SYSTAT does a nested sort. For example, if you select *SEX\$* and *AGE* from the *CHILDREN* data file in that order, SYSTAT first sorts the males and females into two groups and then sorts each group from youngest to oldest.

The input is:

```
USE CHILDREN
SORT SEX$ AGE
LIST
```


The output is:

Case	SEX\$	AGE	N
1	Female	3	8
2	Female	5	1
3	Female	5	5
4	Female	5	10
5	Female	5	13
6	Female	6	4
7	Female	6	14
8	Male	4	3
9	Male	4	11
10	Male	5	12
11	Male	6	2
12	Male	6	6
13	Male	6	9
14	Male	8	7

Within each gender group, SYSTAT arranges the cases so that the values for *AGE* go from smallest to largest.

Example 9

Sorting in Ascending and Descending Order

You can sort data in ascending or descending order. Specify D or A for each variable. For example:

```
USE CHILDREN
SORT SEX$ AGE / D A
LIST
```

sorts the file in descending order for *SEX\$* (males precede females) and then, within each sex, in ascending order for *AGE* (from youngest to oldest).

The output is:

Case	SEX\$	AGE	N
1	Male	4	3
2	Male	4	11
3	Male	5	12
4	Male	6	2
5	Male	6	6
6	Male	6	9
7	Male	8	7
8	Female	3	8
9	Female	5	1
10	Female	5	5
11	Female	5	10
12	Female	5	13
13	Female	6	4
14	Female	6	14

Example 10

Selecting Subset of Cases

Using the *USSTATES* data file, select states in the *Pacific* and *Mountain* divisions that have low rainfall.

The input is:

```
USE USSTATES
SELECT (RAIN < 15) AND (DIVISION$ = 'Pacific' OR,
      DIVISION$ = 'Mountain')
LIST RAIN STATE$
```

The output is:

Case	RAIN	STATE\$
38	11.000	MT
39	11.000	ID
42	10.000	NM
43	7.000	AZ
44	14.000	UT
45	8.000	NV

Example 11

Selecting Cases with equal values of some Variables

You can select cases for which values of some specified pairs of variables are equal. For example, the commands:

```
NEW
INPUT A B C D
2 2 2 2
3 3 3 4
5 5 1 1
6 7 0 1
8 9 1 0
~
SELECT (A=B) AND (B=C)
LIST
```

selects all the cases where the “(A=B) and (B=C)” condition is satisfied i.e. all the cases for which the values of all the variables *A*, *B*, and *C* are equal.

Thus the output is

Case	A	B	C	D
1	2.000	2.000	2.000	2.000
2	3.000	3.000	3.000	4.000

Now the commands:

```
SELECT (A=B) and (B=C) and (C=D)
LIST
```

selects cases where “(A=B) and (B=C) and (C=D)”. Here, you can see that case 2 has not got selected as in the earlier case. In this case the output is

Case	A	B	C	D
1	2.000	2.000	2.000	2.000

However, SELECT A=B=C does not compare A, B, C simultaneously.

The commands:

```
SELECT A=B=C
LIST
```

results in the following:

The first two variables i.e., A and B are compared and if they are equal for a case then the value '1' is being stored; otherwise '0' is stored. And then if the next variable C equals the stored value then that case is selected. Hence, for the above command the output is:

Case	A	B	C	D
3	5.000	5.000	1.000	1.000
4	6.000	7.000	0.000	1.000

Similarly, the case for SELECT A=B=C=D.

First, A and B are compared and if they are equal then '1' is stored; otherwise '0' is stored. Then, if the values of the next variable C equals the stored value (i.e. '0' or '1') then '1' is stored; otherwise '0' is stored for further comparison. Finally, if the last variable D equals the stored value then the case is selected; otherwise not.

Thus for the commands:

```
SELECT A=B=C=D
LIST
```


The output is:

Case	A	B	C	D
3	5.000	5.000	1.000	1.000
4	6.000	7.000	0.000	1.000
5	8.000	9.000	1.000	0.000

Grouping Variables and By Groups

For each case, a grouping variable contains a value that identifies group membership. For example, for the string variable *SEX*, the values might be *Male* and *Female*, or for the numeric variable *SEX*, the codes might be 1 and 2. The values of grouping variables are used to define categories, cells, subpopulations, or groups of cases for:

- Each factor in a frequency table.
- Analyzing where group means are compared, such as *t* tests, analysis of variance, or discriminant analysis.
- Stratifying an analysis—for example, computing descriptive statistics separately for males and females using SYSTAT's By Groups feature.
- Univariate graphical displays, such as bar charts, pie charts, dot plots, and profile displays.
- Symbols or names that label plot points in bivariate and 3-D scatterplots.

The values of grouping variables can be entered and stored as a column in your data file or generated while executing any SYSTAT procedure via the Label, Order and Recode features or the COD, INC, and CUT functions.

This chapter describes how to:

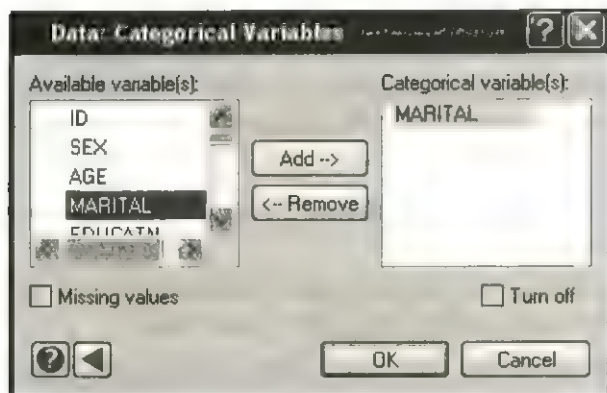
- Specify categorical variables.
- Assign labels.
- Order categories and labels for displays and analyses.
- Recode values of numeric or string variables.
- Specify cutpoints to define intervals along a continuous variable.

- Identify when the value of a numeric or string variable matches one of a set of codes that you list (1 = *yes*, 0 = *no*).
- Define By Groups variables for stratified analyses.

Categorical Variables Dialog Box

To treat values of a numeric or character variable as categories, click CAT on the Status bar, or from the menus choose:

Data
Categorical Variables...



SYSTAT treats each value of the selected variables as a discrete category. Thus, if you identify a quantitative variable as categorical, each unique value appears on the plot scale, the values are equally spaced (for example, if the numbers are 1, 3, and 15, they are spaced as 1, 2, 3), and the minimum and maximum data values are limits for the plot scale.

To set individual variables as categorical, use the CATEGORICAL column in the Variable Editor. Double-click the corresponding cell in the column, and select YES from the dropdown list. You can also set a variable as categorical at the stage of variable definition itself, that is, by checking Categorical in the Variable Properties dialog.

Missing values. Specifies that cases with a missing value for the categorical variable be included as an additional category.

Turn off. Removes any category specifications currently in effect.

Saving category information. By default, when you request data to be saved to a SYSTAT data file, any category information that you set will also be saved. The implication of this is that the next time you open the saved data file and run an analysis or draw a graph involving the saved categorical variables, they will be processed as categorical variables without any explicit declaration. If you do not want this to happen, that is, if you want variables to be treated as categorical only after you declare them as such, then uncheck Save Category variable information to data file in the Data tab of the Edit:Options dialog box.

NCAT Function

Use the NCAT function to know the number of category levels in a variable. The syntax is:

`NCAT (arg1, arg2)`

arg1 is a numeric or string variable name. The function returns the number of category levels in arg 1. arg2 is an optional argument which can take a value of 0 or 1. If you wish to include a missing value as a category, then use 1 as its value. Else use 0 (default).

Value Labels Dialog Box

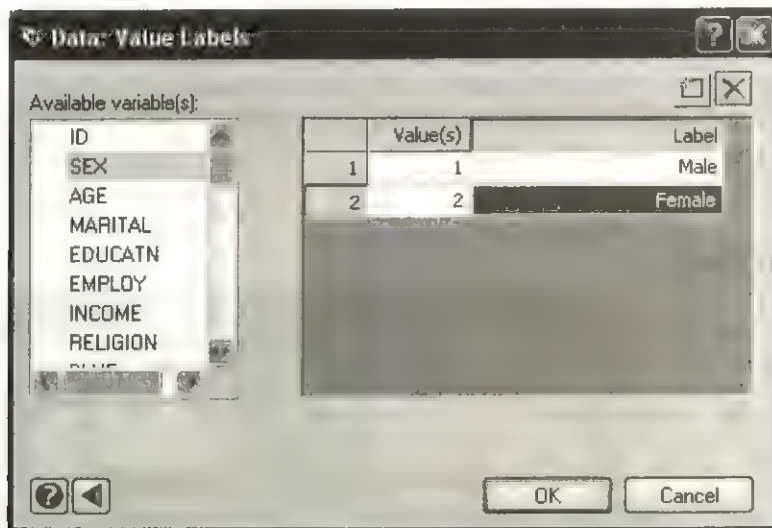
You can use the Value Labels dialog box or the LABEL command to:

- Assign a character name to a value for use as a label in the output.
- Order categories for graphical displays and statistical analyses.
- Assign new labels for string variables.

To open the Value Labels dialog box, from the menus choose:

Data

Value Labels...



In the Value Labels dialog box, select a variable and then specify the values and corresponding labels related to that variable. You can define value labels for any number of variables in this manner by simply selecting that variable and then entering values and labels.

You can also specify value labels for a given variable using the Value Labels column in the Variable Editor.

- Click the Variable Editor tab in the Viewspace.
- Click the desired cell in the Value Labels column.
- Click the ellipsis button. The Value Labels dialog pops up.
- Enter the data values with their desired labels.
- Press OK.

To remove a previous label specification, open the Labels dialog box again and select Turn off.

Displaying value labels in the output. By default, value labels are displayed in the output for all variables for which value labels have been defined; the values themselves

are displayed for other variables. You can control the display of value labels using the Value labels display setting in the Edit:Options dialog or the LDISPLAY command or, from the menus choose:

Edit
 Output Format
 Value Label Display...

The Label menu item will be checked by default. Use the Data setting to suppress the display of value labels, and Both to display value labels as well as data values.

Saving labels with the data. Any value labels that you define will also be saved to the file if you request saving the data after defining the labels. The value label information will be associated with the corresponding variable information and not as a separate variable.

You can also create a variable containing the value labels defined for another variable using the LAB\$ function [LET var\$ = LAB\$(var)], and then save the data file. In this method of saving, however, SYSTAT will not associate the new variable as being the value label equivalent of the old variable.

Use of quotation marks. With the Labels dialog box, you do not need to type quotation marks. With commands, if the name of a category or interval is a number, you can omit the quotation marks.

Category for Missing Codes

If you want to include a category (group) for cases with missing codes, and you want to label it, your option list should include a period (.) for missing numbers or a space surrounded by quotation marks (" " or ') for missing character data. For example,

```
LABEL gnp$ / ' '=Missing, 'D'=Developed, 'U'=Emerging
```

forms three economic categories for the countries in the *OURWORLD* file.

Order of Display Dialog Box

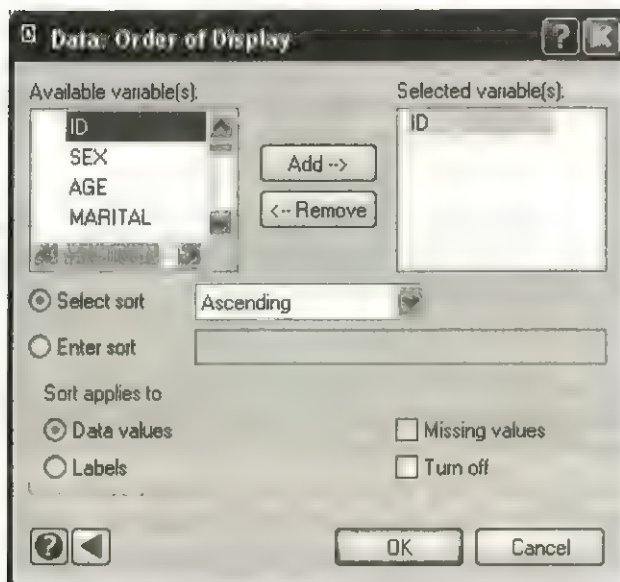
By default, SYSTAT orders numeric category codes or labeled values in the ascending order of their magnitude, and string category codes or labeled strings in the alphabetical order. You can use Order of Display on the Data menu or the ORDER

command to specify how SYSTAT should sort categories or labels for output including table factors, statistical analyses, and graphical displays.

To open the Order of Display dialog box, from the menus choose:

Data

Order of Display...



Select sort. Specify one of the following options for ordering categories:

- **None.** Categories or labels are ordered as SYSTAT first encounters them in the data file.
- **Ascending.** Ascending sort. Numeric category codes or labels are ordered from smallest to largest, and string codes or labels, alphabetically. This is the default.
- **Descending.** Descending sort. Numeric category codes or labels are ordered from largest to smallest, and string codes or labels, backward alphabetically.
- **Ascending frequency or Descending frequency.** Categories or labels are ordered by the frequency of cases within each, placing the category or label with the largest (or smallest) frequency first. Use Ascending frequency for an ascending sort and Descending frequency for a descending sort.

Enter sort. Specifies a custom order for codes or labels. Values must be separated by commas, with string values enclosed in quotation marks (for example, 1, 3, 2, or 'low,' 'high').

Sort applies to. Indicates whether the sort applies to data values or value labels. By default, the sort applies to data values even if value labels are defined. Select Label if you want the sort to be based on value labels.

Missing values. Includes an additional category for missing values if the value of *var* or *var\$* is missing.

Turn off. Returns to the default order.

ORDER is not a filter. For example, if the data for a variable contain six unique values and you identify three of them using the SORT option, then the remaining (unspecified) values are still considered for categorizing and are automatically sorted in the file order when they are shown in the output.

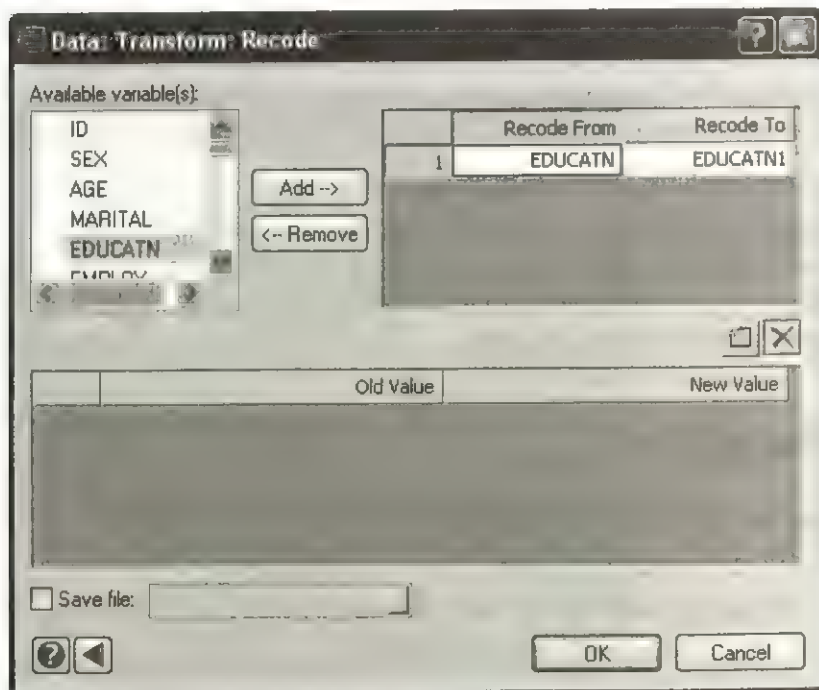
Recode Dialog Box

You can use the Recode dialog box or the RECODE command to:

- Group cases into categories
- Recode the values of existing variables
- Create new variables with recoded values

To recode values of numeric or string variables, from the menus choose:

Data
Transform
Recode...




Select one or more variables in the Available variable(s) list, and press Add to add them to the Recode From column.

Recode From. This column accepts either numeric or string variables one on each row.

Recode To. By default, the values of the selected variable(s) will be replaced by the new values that you specify. To create a new variable with the recoded values of a selected variable, specify a variable name in the corresponding row of the Recode To column.

All the Recode From variables will be recoded based on the old and new values that you specify in the columns provided.

Old Value. Specify the value that you want to recode. You can specify multiple values separated by spaces, commas or the ellipsis notation (...). The ellipsis notation will allow you to specify a range of values starting from a given value upto another given

value. If any value that you enter here does not exist in the selected variable(s), it will be ignored while recoding. So, it is not necessary that you should only enter values that exist in one of the selected variable(s). Do not enclose strings in quotes; the dialog takes care of inserting quotes whenever required. Press the Enter key or the  button to add new rows.

New Value. Specify the desired recoded value; specify a single number or string as applicable. Strings may include spaces. Do not enclose strings in quotes; the dialog takes care of inserting quotes whenever required.

Save file. You can save the file with the recoded values to a SYSTAT data file.

COD, INC and CUT Functions

The Recode dialog allows you to recode single, multiple or a range of values. For specific kinds of recoding, you can also use the functions COD, INC and CUT. With these, you can:

- Recode values of numeric or string grouping variables: COD.
- Identify when the value of a numeric or string variable matches (1 = *yes*, 0 = *no*) one of the specific codes you list: INC.
- Define intervals along a continuous variable: CUT.
- Separate the values of a string variable alphabetically: CUT.

These functions are specified using LET... and IF... THEN LET statements.

Unlike LABEL and ORDER, these functions modify the actual data values. If you save the data after specifying COD, INC, or CUT, the results are stored in the data file.

Value Recoding: COD

Use the COD function to recode values of a grouping variable.

Function	Result
COD(<i>var</i> , <i>num1</i> , <i>num2</i> ,...)	Values of <i>var</i> are replaced: <i>num1</i> becomes 1, <i>num2</i> becomes 2, etc.
COD(<i>var</i> \$,' <i>text1</i> ', ' <i>text2</i> ',...)	Values of <i>var</i> \$ are replaced: <i>text1</i> becomes 1, <i>text2</i> becomes 2, etc.

The COD function always produces consecutive integers (starting with 1) that correspond, respectively, to the original codes specified in the statement. Periods (SYSTAT's marker for missing data) are stored in the recoded variable for every case whose original value is not among the values specified.

Comparing Values: INC

The INC (for included) function compares each value of a numeric or string variable with a list of values that you specify. If a match is found, SYSTAT returns a 1; if a match is not found, a 0.

Function	Result
INC(<i>var</i> , <i>num1</i> , <i>num2</i> ,...)	A 1 (true) if the value in <i>var</i> matches one of the numbers <i>num1</i> , <i>num2</i> , ...; 0 (false) otherwise.
INC(<i>var</i> \$,' <i>text1</i> ',' <i>text2</i> ',...)	A 1 (true) if the value in <i>var</i> \$ matches <i>text1</i> , <i>text2</i> , ...; 0 (false) otherwise.

Defining Intervals: CUT

Use CUT to define intervals on a quantitative (continuous) variable or to divide the values of a string variable alphabetically.

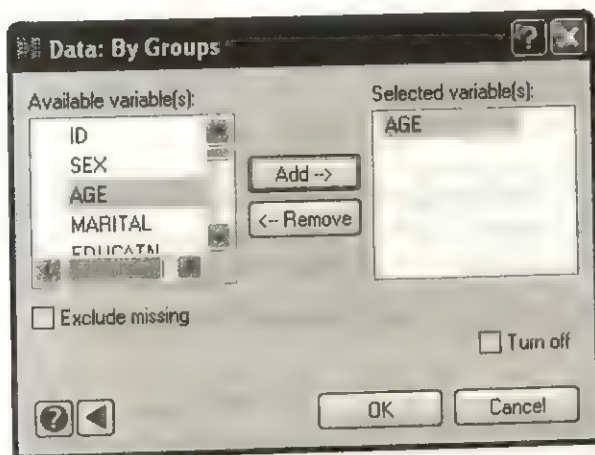
Function	Result
CUT(<i>var</i> , <i>num1</i> , <i>num2</i> ,...)	Use to define intervals along a continuous variable. Values in <i>var</i> less than or equal to <i>num1</i> get value 1. Values greater than <i>num1</i> and less than or equal to <i>num2</i> get value 2, etc.
CUT(<i>var</i> \$,' <i>text1</i> ',' <i>text2</i> ',...)	Use to group variables alphabetically. Values in <i>var</i> \$ less than <i>text1</i> (alphabetically), including <i>text1</i> , get value 1. Values between <i>text1</i> and <i>text2</i> , including <i>text2</i> , get value 2, etc.

By Groups Dialog Box

You can use By Groups on the Data menu or the BY command to request separate results for each level of one or more grouping variables. For example, you could obtain separate graphs and statistical analyses for male children, female children, male teens, female teens, and so on. Here, *SEX* and *AGE*\$ (categorized as child, teen, etc.) would be the grouping variables.

To open the By Groups dialog box, click BY on the status bar or from the menus choose:

Data
By Groups...



You can select up to 10 grouping variables. After specifying grouping variables, run your analyses as normal; all subsequent SYSTAT procedures and graphs perform analyses separately for each unique combination of the group codes.

Exclude missing. Excludes categories defined by missing values from the analysis. If this option is not selected, a separate category is created for cases with missing values for the grouping variable, and a separate analysis is carried out for them.

Turn off. Turns off By Groups so that subsequent analyses are performed across all cases. You can also turn off By Groups by quitting SYSTAT or opening a new data file.

By Groups versus Plot Groups

When By Groups is used with a graphical display, each group (or each unique combination of the grouping variables) is displayed in a separate chart, and the scale is set using only the values within the group. For example, if *SEX* is the grouping variable, separate charts are produced for males and females. The scale for each chart is defined by the maximum and minimum values within each gender.

However, for many graphical displays, a Group option is available. With this option, displays for all subgroups appear on a single chart and share a common scale.

To obtain separate charts that share a common scale, use the minimum and maximum values of the complete sample to define the scale, rather than using the minimum and maximum values within each particular group.

Using Commands

To treat variables as categorical, use

```
CATEGORY varlist / ADD or REPLACE MISS
```

ADD will add *varlist* to the existing category set and is the default setting. REPLACE will replace the existing category set by *varlist*. The optional MISS will include missing values as a category in each of the category variables.

To display a list of the currently declared categorical variables, use

```
CATEGORY / ?
```

To turn off category declarations, execute

```
CATEGORY varlist / OFF
```

If you do not specify *varlist*, all category declarations are turned off.

Use this syntax for labeling numeric values:

```
LABEL varlist / n1='text1', n2='text2',  
or  
n1,n2,...='text1',  
n3,n4,...='text2',  
or  
n1 .. n2='text1',  
n3 .. n4='text2',
```

For string values, specify:

```
LABEL varlist$ / 'oldtext1'=newtext1,  
'oldtext2'=newtext2,
```

You can also use VALLABEL instead of LABEL.

Options for LABEL statements include:

<code>n1='text1', n2='text2', ...</code>	The values <i>n1</i> , <i>n2</i> , ... in the data file are labeled <i>text1</i> , <i>text2</i> , ..., respectively.
<code>n1,n2, ...='text1', n3,n4, ...='text2', ...</code>	The values <i>n1</i> , <i>n2</i> , ... are labeled <i>text1</i> , the values <i>n3</i> , <i>n4</i> , ... are labeled <i>text2</i> , etc. (The values <i>ni</i> do not have to be consecutive values or specified in order).
<code>n1 .. n2='text1' n3 .. n4='text2', ...</code>	The values <i>n1</i> and <i>n2</i> and all the values between them are labeled <i>text1</i> ; the values from <i>n3</i> to <i>n4</i> are labeled <i>text2</i> ; and so on. If <i>n1=n3</i> , it will be considered in the interval where it is the upper limit.
<code>'oldtext1'=newtext1 'oldtext2'=newtext2, ...</code>	Assign new names to string values.

To order categories or value labels:

```
ORDER varlist or varlist$ / MISS,
                             DATA or LABEL,
                             SORT=option
```

where option is NONE, ASC, DESC, FASC, FDES, numlist, or charvallist. By default, the sort is based on data values unless you specify LABEL. Specify ORDER var (or var\$) to return to the default order.

Use this syntax for recoding numeric values:

```
RECODE varlist / n1=m1, n2=m2,
                  or
                  n1,n2,...=m1,
                  n3,n4,... =m2,
                  or
                  ..n0=m0
                  n1 .. n2=m1,
                  n3 .. n4=m2,...,
                  nk ..      =mk,...,
```

The second set of options allows you to put two or more groups into a larger group. The third set allows you to discretize a continuous variable, that is, define intervals along a continuous variable.

For string values, specify:

```
RECODE varlist$ / 'oldtext1'=newtext1,
                  'oldtext2'=newtext2,...
```


To store recoded numeric values into a new numeric variable, issue:

```
RECODE newvar = oldvar /  n1=m1, n2=m2,
                        or
                        n1,n2,...=m1,
                        n3,n4,...=m2,
                        or
                        .. n0=m0
                        n1 .. n2=m1,
                        n3 .. n4=m2,...,
                        nk ..      =mk
```

To store recoded numeric values into a new string variable, issue:

```
RECODE newvar$ = oldvar /  n1='text1', n2='text2',...
                        or
                        n1,n2,...='text1',
                        n3,n4,...='text2',...
                        or
                        .. n0='text0'
                        n1 .. n2='text1',
                        n3 .. n4='text2',...,
                        nk ..      ='textk'
```

To store recoded string values into a new numeric variable, issue:

```
RECODE newvar = oldvar$ /  'text1'=n1,
                        'text2'= n2,...
```

To store recoded string values into a new string variable, issue:

```
RECODE newvar$ = oldvar$ /  'oldtext1'=newtext1,
                        'oldtext2'= newtext2,...
```

In each of the above RECODE statements, value(s) given in the left side of an option are recoded to the value in the right side of the option.

To request separate results for each level of one or more grouping variables, use the BY command before you request a graph or analysis:

```
BY grpvar1, grpvar2, ...
```

For example, if the variable *SEX* has code 1 for males and code 2 for females, and *CITY\$* has the codes Chicago, New York, and Los Angeles, specify

```
BY sex, city$
```

to form six groups (Chicago males, Chicago females, New York males, New York females, and so on).

- The seven codes of EDUCATN for educational level are collapsed into four categories of the variable EDUCATN\$ with the assigned names.

```
RECODE AGE$=AGE /    .. 29 ='18 to 29',
                    30 .. 45 ='30 to 45',
                    46 .. 60 ='46 to 60',
                    60 ..    ='Over 60'
or
RECODE AGE$=AGE /    .. 29 ='18 to 29',
                    .. 45 ='30 to 45',
                    .. 60 ='46 to 60',
                    60 ..    ='Over 60'
```

- The age of each subject is used to assign the person to one of four age groups. That is, the ages 29, 45, and 60 are used as cutpoints along the continuous variable *AGE* to form four intervals. The second specification is a shortcut and works when the intervals are identified in the same order as they fall across the range of the distribution. SYSTAT forms the category as it encounters each specification. Thus, if we specify:

```
RECODE AGE$ = AGE / ..30 ='label1', 30..='label2'
```

- SYSTAT assigns 30-year-olds to the first group and anyone 30.00001 or older to the next group.

```
LABEL SEX$ / 'M'=Male, 'F'=Female
```

- The character codes *M* and *F* define two subpopulations—they are labeled *Male* and *Female*, respectively.

```
LABEL SEX$ / 'M'=Male, 'F'=Female, ' '= Missing
```

- When this LABEL statement is specified, missing values, corresponding to those subjects who did not report their sex, will be labeled as "Missing" in any output that includes missing values.

```
SELECT DOSE$ <> 'none'
ORDER DOSE$ / SORT='high', 'medium', 'low'
```

- For character codes, you may need to eliminate a category and/or specify a specific order. Here, SELECT is used to delete the subjects who had no drug (*none*) and order the doses from high to low for use in graphs and analyses.

RECODE in some of the above examples and SELECT both select a subset of cases; however, the subset identified via SELECT applies to all subsequent displays and analyses; that for RECODE applies only to those subsequent procedures where the same recoded variables are specified.

Example 2

Labeling Multiple Variables

If you have several variables with the same codes, you can specify grouping information for them in one statement using commands. For example, if your data file has the same doses (6.25, 12.5, and 25) for three drugs (where each drug is a variable), specify labels as follows:

```
LABEL DRUG_A DRUG_B DRUG_C / 6.25='low dose',
                               12.5='medium dose',
                               25='high dose'
```

Alternatively, if the three drug variables are stored consecutively in the data file, use the ellipsis notation (...):

```
LABEL DRUG_A .. DRUG_C / 6.25='low dose',
                          12.5='medium dose',
                          25='high dose'
```

Or, as an example using character data, consider a study about smoking cessation in which researchers questioned subjects about their desire to have a cigarette in different situations (after eating, at a party, and after a stressful event). To each question, the subjects responded "always," "sometimes," or "never." For quick data entry, you could name the questions 1 through 10 and enter the responses as *a*, *s*, and *n*. Then specify

```
LABEL Q1$ .. Q10$ / 'a'=always 's'=sometimes 'n'=never
```

to tell SYSTAT to use the full response to label the output.

Example 3

Collapsing Categories

If you assign the same label to two codes (or groups of codes), the codes form a single category. The *CANCER* data file presents data from a breast cancer study in which 764 women have tumors classified in one of four categories. Two of the categories describe malignant tumors, and the other two, benign tumors. To find out how the overall

malignant or benign status relates to survival, the RECODE command is used to collapse the four categories into two:

```
XTAB
  USE CANCER
  FREQ NUMBER
  RECODE TUMOR$ / 'MinMalig' 'MaxMalig' = Malignant,
                 'MinBengn' 'MaxBengn' = Benign
  PLENGTH NONE / FREQ CHISQ
  TABULATE SURVIVE$ * TUMOR$
```

The output is:

Case frequencies determined by value of variable NUMBER

Counts

SURVIVE\$(rows) by TUMOR\$(columns)

	Benign	Malignant	Total
Alive	323	231	554
Dead	97	113	210
Total	420	344	764

Chi-square tests of association for SURVIVE\$ and TUMOR\$

Test Statistic	Value	df	p-value
Pearson Chi-square	9.026	1.000	0.003

Number of Valid Cases: 72

The proportion of women with benign tumors who remained alive at the end of the study is significantly greater than the proportion with malignant tumors ($p\text{-value} = 0.003$).

Example 4

Recoding Numeric Variables

Suppose the variable *OCCUPATN* contains 30 or 40 numeric occupation codes including: 21 (bricklayer), 32 (carpenter), 53 (day laborer), and 65 (plumber). For a particular analysis, you need to study these four jobs and want to recode the values as 1 (day laborer), 2 (carpenter), 3 (plumber), and 4 (bricklayer). Use either the RECODE command as illustrated earlier or the COD function to do this:

```
RECODE NEW_JOB = OCCUPATN / 53=1, 32=2, 65=3, 21=4
or
LET NEW_JOB = COD (OCCUPATN, 53, 32, 65, 21)
```


The values of *OCCUPATN* and the corresponding values of *NEW_JOB* are shown below:

<i>OCCUPATN</i>	<i>NEW_JOB</i>
53	1
32	2
65	3
21	4
All else	.

Example 5

Recoding Characters to Numbers

The COD function also works for string variables. Suppose the string variable *OCCUPATN\$* has 30 or 40 string codes. The values of *NEW_JOB* will be 1, 2, 3, and 4 if we specify the following:

```
LET NEW_JOB = COD (OCCUPATN$, 'day laborer', 'carpenter',
                    'plumber', 'brick layer')
```

The equivalent RECODE command is:

```
RECODE NEW_JOB= OCCUPATN$/'day laborer'=1,
                      'carpenter'=2,
                      'plumber'=3,
                      'brick layer'=4
```

Always remember to use quotation marks to surround all string codes that you specify in the argument of the COD function or the option of the RECODE command.

Example 6

Matching Values

Valid INC statements include:

```
LET DRIVER = INC(JOB, 63, 47, 85)
LET DRIVER = INC(JOB$, 'truck driver', 'bus driver',
                  'cab driver')
```

In the first statement, if *JOB* contains a 63, 47, or 85, *DRIVER* will contain a 1. *DRIVER* contains a 0 for all other values of *JOB*. In the second statement, if *JOB\$* contains 'truck driver,' 'bus driver,' or 'cab driver,' the new variable *DRIVER* will contain a 1. All other values for *JOB\$* receive a 0 for *DRIVER*.

Example 7

Categorizing Variables

Sometimes you need to categorize a quantitative variable, such as *AGE*. For example, this statement uses each subject's age to assign him or her to one of four age groups:

```
LET AGE_GRP = CUT (AGE, 12, 19, 64)
```

The values of *AGE* and the corresponding values of the new variable *AGE_GRP* are shown below:

<i>AGE</i>	<i>AGE_GRP</i>
0 < ages <= 12	1
12 < ages <= 19	2
19 < ages <= 64	3
64 < ages	4

The numbers you list in the CUT function (12, 19, and 64) are the upper limits of the intervals. For example, a 64-year-old subject falls into the third category, and someone who reports their age as 64.01 is in the fourth category. Note that n cutpoints define $n + 1$ intervals, so you always have one more interval than the number of cutpoints.

Alternatively, you could use RECODE to do the same task as the above CUT function:

```
RECODE AGE$ = AGE / .. 12 = 'child',
                  .. 19 = 'teen',
                  .. 64 = 'adult',
                  64 .. = 'senior'
```

Example 8

Alphabetical Intervals

The CUT function also divides string variables alphabetically into intervals. Suppose the variable *NAME\$* contains the following values:

Richards, Carson, Young, Stern, Parker, Buck, Smith, Martin, Howe

The CUT function:

```
LET GROUP = CUT(NAME$, 'Howe', 'Richards')
```


assigns codes alphabetically to the variable *GROUP*:

<i>NAMES</i>	<i>GROUP</i>
Richards	2
Carson	1
Young	3
Stern	3
Parker	2
Buck	1
Smith	3
Martin	2
Howe	1

The new variable *GROUP* contains a 1 for Howe and the names preceding Howe alphabetically, a 2 if the name falls between Howe and Richards (including Richards), and a 3 if it follows Richards.

Example 9 ***Use of NCAT Function***

This function returns a numeric value which gives the number of categories in a given data variable. Consider the example:

```
USE AFIFI
X = NCAT (DRUG)
PRINT 'The no of categories in drug is', x
PRINT 'The no of categories in disease is', NCAT (DISEASE)
```

In the output, the number of categories in drug and disease are displayed as follows.

```
The no of categories in drug is 4.000
The no of categories in disease is 3.000
```


Special Topics in Data Management

The BASIC procedure in SYSTAT is a global feature to facilitate transformation and complex data management tasks requiring a programming language. This procedure allows you to:

- Read ASCII text files with multiple lines per case, multiple cases per record, records of unequal length or with a fixed format.
- Enter data case by case at the command prompt.
- Enter or read matrices with or without a diagonal and specify their type as correlation, covariance, similarity, dissimilarity, and so on.
- Delete cases that meet or fail to meet a specified condition.
- List data without variable names or case numbers.

For more complicated programming tasks, you can:

- Use arrays (subscripted variables).
- Use FOR... NEXT loops.
- Use WHILE.. ENDWHILE loops.
- Include ELSE, PRINT, and DELETE in IF...THEN statements.
- Specify commands that operate on subgroups of cases (beginning and end of group and file).

This chapter presents applications of SYSTAT transformation commands. While there are simpler ways to accomplish many of these tasks, the programs in this chapter are selected to illustrate the full range of SYSTAT's capabilities.

Data Entry and BASIC

BASIC in SYSTAT provides more flexibility for reading ASCII data files than the IMPORT command. You can read files with:

- More than one line of data per case.
- Fixed-format.
- Special data structures, such as multiple cases per line.
- Varying numbers of variables per case.

Here is an overview of the commands for reading a text file:

```
GET input filename
INPUT varlist
ESAVE output filename
```

Input filename is the name of the ASCII file with a *.DAT* extension to be imported; *varlist* is a list of variable names; and *output filename* is the name of the SYSTAT file that is created. SYSTAT uses the number of variable names you list to determine how many values to read for each case.

Entering data. Rather than use a GET statement to read an existing file, you can type small data sets on the Interactive command tab as follows:

```
INPUT varlist
      [type data here]
```

Type a tilde (~) or ENDINPUT to end data input.

To save the file use the command:

```
ESAVE output filename
```

Free- or fixed-format input. You can use free-format or fixed-format to input data with BASIC commands or to format data in an ASCII text file.

Free-Format Input

Free-format input is easier to use than fixed-format because it reads delimited data.

- Each data value is separated by a tab, a comma, and/or one or more spaces.
- Each new case begins on a new line. A single case can extend over several lines, but the next case must start on a new line. (If you have several cases per record, use a backslash after the varlist.)
- Character values that contain blanks, commas, or special characters are enclosed in single (' ') or double (" ") quotation marks.
- Missing numeric data are recorded as a period (.), and missing character data are recorded as a blank enclosed in single (' ') or double (" ") quotation marks. They can also be indicated by typing two commas (, ,) with no characters between them.

The data file should be ASCII text without nonprinting characters such as page breaks, margin indicators, or control characters. The data can be aligned or unaligned:

Sloan	male	31	158.5		Sloan male 31 158.5
'Mike Johnson'	male	45	165	or	'Mike Johnson' male
Smythe	female	37	126.5		45 165
					Smythe female 37
					126.5

SYSTAT counts the number of variable names listed in the INPUT command to determine how many data values to read for each case. After reading the specified number of values, SYSTAT moves to the next line to begin reading the next case.

Use INPUT to name the variables in the order they are read into SYSTAT:

```
INPUT varlist
```

where *varlist* is a list of variable names. You can identify a range of variables in *varlist* using subscript notation, such as:

```
INPUT var(1)..var(10)    or    INPUT var(1..10)
```

Variable names can be up to 256 characters long. Names of string variables must end with a dollar sign (\$). Subscripts can be used for numeric or string variable names (for example, *QUESTION(1),...,QUESTION(35)*) as long as the total length of the variable name does not exceed 256 characters.

Example 1

Inputting Data at the Command Prompt

You can create a data file by typing data directly in the interactive tab of the command space. For example, to create a SYSTAT data file called *MYFILE*, type:

```
INPUT A B C
```

Now enter data, one case to a line, pressing the Enter key at the end of each line:

```
1 2 3
4 5 6
7 8 9
~
ESAVE myfile
```

The character, a tilde (~), tells SYSTAT to end data input.

Typing data at the command prompt is particularly useful if you are creating a command file and want to include the data in that file rather than in a separate data file.

Example 2

Reading an ASCII Text File: GET

Suppose you want to read an ASCII text file named *INFILE.DAT* that has the following data:

```
1 2 3
4 5 6
7 8 9
```

However, you then want to save this file as a SYSTAT file named *MYFILE1.SYZ* that has three variables named *A*, *B*, and *C*. The input is as follows:

```
GET INFILE
INPUT A B C
ESAVE MYFILE1
```


Example 3

Reading Multiple Lines per Case

To demonstrate how to read ASCII files with multiple lines per case, we use the *MINIWRLD.DAT* file. This file contains 15 variables for each country and has two lines of data per case. The variables are:

- Name of the country.
- Population in 1990.
- Percentage of the population living in cities.
- Number of births per 1000 people per year.
- Number of infants (per 1000 live births) who die during their first year.
- Gross domestic product per capita.
- Average expenditure for education (per person).
- Average expenditure for health (per person).
- Average expenditure for military (per person).
- Type of government.
- Years of life expectancy for males.
- Years of life expectancy for females.
- Percentage of population who can read.
- Type of country.
- Ratio of birth to death rates.

The following illustration shows the file in a text editor.

France	56.358	73	14	6	14542.657	648.069	728.249	432.780	Democracy
73 82	99.00	Europe		1.556					
Greece	10.028	65	11	10	5614.184	115.000	158.700	260.400	Democracy
75 80	95.00	Europe		1.222					
Switzerland	6.742	58	12	5	17723.499	853.538	1209.077	330.769	Democracy
75 83	99.00	Europe		1.333					
Spain	39.269	91	11	6	10153.121	166.675	224.923	117.964	Democracy
75 82	97.00	Europe		1.375					
UK	57.366	76	14	7	14259.401	474.488	479.064	447.473	Democracy
73 79	99.00	Europe		1.273					
Hungary	10.569	54	12	15	6112.397	234.340	196.792	146.321	OneParty
67 75	99.00	Europe		0.923					
Iraq	18.782	68	46	67	1863.509	87.875	19.000	760.000	OneParty
66 68	55.00	Islamic		6.571					
Pakistan	114.649	28	43	110	376.801	7.655	0.697	23.268	Military
56 57	26.00	Islamic		3.071					
Ethiopia	51.667	14	45	116	127.742	4.943	1.185	10.137	Military
49 52	55.20	Islamic		3.000					
Afghanistan	15.862	16	44	154	189.128	1.000	.	.	OneParty
47 46	12.00	Islamic		2.444					
Brazil	152.505	68	26	69	2472.049	59.449	22.729	16.260	Democracy
62 68	76.00	'New World'		3.714					
Canada	26.538	76	14	7	19353.214	1052.813	936.211	316.797	Democracy
74 81	99.00	'New World'		2.000					
Venezuela	19.698	76	28	27	2944.446	193.989	78.652	45.562	Democracy
71 77	85.60	'New World'		7.000					
El Salvador	5.310	39	34	49	1035.808	17.647	7.647	29.020	Military
62 68	65.00	'New World'		4.857					

To store these data in the SYSTAT file called *MINIWRLD.SYZ*, the input is:

```
GET MINIWRLD
INPUT COUNTRY$ POP_1990 URBAN BIRTH_RT BABYMORT,
      GDP_CAP EDUC HEALTH MIL GOV$ LIFEEXPM,
      LIFEEXPF LITERACY GROUP$ B_TO_D
ESAVE MINIWRLD
```

Example 4

Reading Records of Unequal Length

For each new case, SYSTAT reads as many data values as are named in the INPUT statement, one value per variable, even if it has to read several lines of data to do so. The following example illustrates what happens when the rows have records that vary in length. The data in the text file *MYDATA1.DAT* are:

10	20	30	40
50	60		
70	80		
90	100	110	120

To create a SYSTAT file named *MYFILE2.SYZ*, type:

```
GET MYDATA1
INPUT A B C D
ESAVE MYFILE2
```

To view the contents of the SYSTAT file, type:

```
LIST
```

The output is:

Case	A	B	C	D
1	10.000	20.000	30.000	40.000
2	50.000	60.000	70.000	80.000
3	90.000	100.000	110.000	120.000

Line by line, this is how SYSTAT processes the file:

- **Case 1.** SYSTAT reads 10, 20, 30, and 40 as the first case of the new data file.
- **Case 2.** Next, SYSTAT reads two data values from the second line (50 and 60) as variables *a* and *b* for case 2. It still must fill variables *c* and *d*, so it continues to the next line and reads 70 and 80.
- **Case 3.** Finally, SYSTAT reads 90, 100, 110, and 120 as case 3 of the file.

Example 5 Coding Missing Values

SYSTAT uses a period to represent missing numeric data and a space to represent missing character data.

- Code missing numeric data as periods in your data. Do not code missing numeric data as characters such as *na*, *m*, ***, or *?* because SYSTAT does not read character data into a numeric field. If missing numeric data are left as blank spaces, SYSTAT moves the next value in the file where the missing value should be. All subsequent data are also moved. (See the following example.)
- Code missing character data as a blank space enclosed in single (') or double quotation marks ("").

This example demonstrates what happens when you do not code missing values as periods. The values in the *MYDATA2.DAT* text file are:

```
100    200    300
400          600
700    800    900
```

To create a SYSTAT file named *NEWFILE.SYZ*, use the following input:

```
GET MYDATA2
INPUT A B C
ESAVE NEWFILE
```

The SYSTAT file *NEWFILE.SYZ* does not contain the correct number of cases or values, as is shown here:

Case	A	B	C
1	100.000	200.000	300.000
2	400.000	600.000	700.000

Instead of three cases, SYSTAT produces two, omitting 800 and 900. SYSTAT reads the first line of data correctly, but treats the missing value in the second line as a space delimiter separating the values 400 and 600. Therefore, SYSTAT reads 600 as variable *b*. It completes case 2 by reading 700, the first value of the next line. SYSTAT is ready to start a new case, so it moves to the next line of original data. There are no more lines of data to read, so SYSTAT closes the file.

To read these data correctly, use a period to mark where data are missing. Data in the *MYDATA2.DAT* file should be as follows:

```
100    200    300
400    .    600
700    800    900
```


Example 6

Reading Multiple Cases per Record

If you want SYSTAT to read more than one case per line, append a backslash (\) to your INPUT statement. The backslash forces SYSTAT to use all the data in the row, even if it has to start a new case. It also forces SYSTAT to start a new case whenever it starts reading a new row of values. Without a backslash, SYSTAT ignores extra values in a row and moves to the next line to find data for the next case.

This example shows how to use the backslash to read data where you have more than one case per line of original data. Data in the text file *MYDATA3.DAT* are:

Tom	23	Jerry	51	Marilyn	50	Lynne	18
Mark	22	Andrew	8	Henry	70	Chris	23

To create the SYSTAT file *MULTIPLE.SYZ*, type:

```
GET MYDATA3
INPUT NAME$ AGE \
ESAVE MULTIPLE
```

With the backslash, SYSTAT reads the entire line of original data even though each line fills up four cases in the SYSTAT file. Without the backslash, SYSTAT reads only the first two values from each line and generates only two cases. To view the contents of the SYSTAT file, type:

```
LIST
```

The listing follows:

Case	NAME\$	AGE
1	Tom	23.000
2	Jerry	51.000
3	Marilyn	50.000
4	Lynne	18.000
5	Mark	22.000
6	Andrew	8.000
7	Henry	70.000
8	Chris	23.000

Example 7**Reading Incomplete Records: Backslash (\)**

This example shows how to use the backslash to read records that do not have an equal number of values per case. The data in the text file *MYDATA4.DAT* are:

```
1      2      3
4      5      6      7
8      9
```

To create the SYSTAT file *UNEQUAL.SYZ*, the input follows:

```
GET MYDATA4
INPUT A B C D \
ESAVE UNEQUAL
```

The resulting SYSTAT data file looks like this:

Case	A	B	C	D
1	1.000	2.000	3.000	.
2	4.000	5.000	6.000	7.000
3	8.000	9.000	.	.

Fixed-Format Input

With fixed-format input, you tell SYSTAT exactly where the values for each variable are located in the data records. Values for a variable must be in the same place for every record.

A fixed-format input statement consists of two parts. The first part provides the variable names that will appear in the SYSTAT file. The second part contains the format that tells SYSTAT where to read values for each variable. The number of items specified in the format must match the number of input variables. Enclose the variable names and the input format in separate sets of parentheses:

```
INPUT (age,sex$,income) (#3,$6,#8)
```

SYSTAT checks the number of items you specify in the format against the number of variables. If they do not match, an error message is displayed.

Formats

Formats control a pointer that tells SYSTAT where to read the next variable value. These formats specify the location and width of fields that contain values. Leading and trailing blanks are ignored for numeric data. All characters within the formatted field, except leading blanks, are read into a character value. Character strings are left-justified.

Format items for fixed-format input include the following:

#n	Read a numeric value in the next n columns.
\$n	Read a character value in the next n columns.
>	Move the pointer one column to the right.
<	Move the pointer one column to the left.
^n	Move the pointer to column n .
/	Move the pointer to the first column on the next record.
%n	Move the pointer to the first column on the n^{th} record.
n*r	Repeat r n times, where r is any of the above.

Some examples follow:

>>>	Move the pointer three columns to the right.
3*>	Move the pointer three columns to the right.
^10	Move to column 10 of the current record.
#4	Read the numeric value in the next four columns beginning at the current column.
\$5	Read the character value in the next five columns beginning at the current column.
^3	Move the pointer to column 3.
>>>>>	Move the pointer five columns to the right.
5*>	Move the pointer five columns to the right.
%2	Move the pointer to column 1 of the second record. (You cannot move back to an earlier record.)
/	Move the pointer to column 1 of the next record.
//	Move the pointer to column 1 two records ahead. (For example, if you are starting on the first record, %3 and // mean the same thing.)
#3	Read the numeric value in the next three columns beginning at the current column.
2*\$3	Read one character value in three columns and then another in the next three columns.

You can use a backslash (\) to keep the pointer at the current position—rather than move to the next row—to begin the next case. This feature allows you to read files with multiple cases per record.

The following points are to be noted when using the backslash (\):

- The backslash (\) cannot be used with the frontslash (/) in the same INPUT formatted statement since, together, they don't give any meaningful structure.
- The backslash (\) cannot be used with the % option in the same INPUT formatted statement since, together, they don't give any meaningful structure.

Note: Use % and ^ rather than / and >, so that you know precisely which record and column you are on. Furthermore, if you have seven records per observation, you need a %7 at the end of your format to ensure that the pointer is positioned correctly for the next observation, even if you read nothing from the seventh field.

Example 8

Fixed Format

Suppose that you have an ASCII file *TESTDATA.DAT* like the one below:

```

0           1           2
12345678901234567890123456
-----
1232 BILLY 0 1 1 1 0 BACDD
BCEAD
7384 SUSAN 1 1 0 1 1 BDAEA
DDEAE
2837 TIM 1 1 1 0 1 CBADE
DDBCA
7484 TOM 0 0 1 0 1 BCDEC
AAEDC
5678 WAYNE 1 1 0 1 0 ADEAA
DACBB

```

The two lines at the top of the example help you count columns, while the first variable in the file is a four-column ID number. The second is the first name of a student. The next five variables are answers to true–false questions and are separated by spaces. The last five variables on the first line are answers to multiple-choice questions and are not delimited. The variables on the second line are answers to five more multiple-choice questions.

SYSTAT reads the data into a SYSTAT file called *TEST.SYZ*. Because the INPUT statement takes up more than one line, a comma is used to continue it onto the next line:

```

GET TESTDATA
INPUT (ID,NAME$, Q(1) .. Q(5), Q$(6) .. Q$(15)),
      (#4,$6,5*#2,>,5*$1,%2,5*$1)
ESAVE TEST

```


Here is how each variable is read by its format description:

- #4 Read numeric value from first four columns into variable ID.
- \$6 Read character value from next six columns into *NAME\$*.
- 5*#2 Read five consecutive two-column numeric values into *Q(1)*, *Q(2)*, *Q(3)*, *Q(4)*, and *Q(5)*.
- > Move pointer one space to the right.
- 5*\$1 Read five consecutive one-column character values into *Q\$(6)*, *Q\$(7)*, *Q\$(8)*, *Q\$(9)*, and *Q\$(10)*.
- %2 Move pointer to second line of input record.
- 5*\$1 Read five consecutive one-column character values into *Q\$(11)*, *Q\$(12)*, *Q\$(13)*, *Q\$(14)*, and *Q\$(15)*.

Example 9

Inputting Fixed Format Data with Backslash (\)

This example tells us how to read files with multiple cases per record using backslash with a specified format. For example, consider the following input:

```
NEW
INPUT ( A B C \ ) (#4, #3, #2)
123456789642
9753146281
~
```

Then it will read the data as follows:

A	B	C
1234	567	89
642	.	.
9753	146	28
1	.	.

Entering Triangular Matrices

Data in ASCII files take the cases-by-variables (rectangular) form by default. SYSTAT enables you to enter triangular matrices, such as those produced by CORR.

Use the TYPE command to indicate what type of matrix you are entering. Valid types are RECTANGULAR, SSCP, COVARIANCE, CORR, DISSIMILARITY, and SIMILARITY. The default is RECTANGULAR.

COVARIANCE designates a covariance matrix and CORR, a correlation matrix. When CORR outputs a triangular matrix to a SYSTAT file, it automatically sets the

type. If you LIST a triangular matrix, the upper triangular portion contains missing values, since the matrices are symmetric and only half the values are needed by the statistical routines.

Example 10

Covariance Matrices

This example reads a covariance matrix from an ASCII file named *MYDATA5.DAT* and saves it in a SYSTAT file named *TURTLE.SYZ*. *MYDATA5* contains the following data records:

```
451.39
271.17 171.73
168.70 103.29 66.65
```

To create the SYSTAT file, the input is:

```
GET MYDATA5
TYPE COVARIANCE
INPUT LENGTH, WIDTH, HEIGHT
ESAVE TURTLE
```

Example 11

Matrices with Missing Diagonals: DIAGONAL

For some types of data, the values on the diagonal are undefined or constant. In these cases, you can input only the values below the diagonal and leave the diagonal missing.

Use *DIAGONAL ABSENT* to signal that the diagonal values are missing. If you do not use this command, *DIAGONAL PRESENT* is assumed. If you input a correlation matrix with *DIAGONAL ABSENT*, SYSTAT sets the diagonal elements to 1.0; otherwise, the diagonal elements are set to missing values.

The following example reads a correlation matrix with no diagonal elements. The matrix is a text file named *MYCORR.DAT* that looks like this:

```
.96  
.90 .91  
.72 .11 .99  
.09 .42 .89 .98  
.59 .51 .73 .90 .93  
.88 .79 .52 .82 .89 .97
```

To store this matrix as the SYSTAT file *COLORS.SYZ*, type:

```
GET MYCORR  
TYPE CORR  
DIAGONAL ABSENT  
INPUT RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET  
ESAVE COLORS
```

Notice that there are only six rows and columns to fill seven variables. The diagonal elements are set to 1.0.

Troubleshooting ASCII Files

You must be sure that your ASCII file does not contain any nonprinting ASCII characters, such as page breaks, control characters, and column markers. SYSTAT can read numbers, alphabetical and keyboard characters, delimiters (spaces, commas, or tabs that separate consecutive values from each other), and carriage returns.

Numeric fields must contain only numeric data; therefore, exclude variable labels or column headings when using GET to read from an ASCII file. Note that SYSTAT can import ASCII data files containing variable names with Open on the File menu.

Most text editors, such as Microsoft Word and WordPerfect, can save files in an ASCII text format without nonprinting characters. If you do not know whether your file is in an ASCII format, use the TYPE command in DOS to view it. If you see any strange characters, your file was not saved in ASCII format, and SYSTAT will not be able to read it.

Errors and Error Messages

The following are some of the error messages encountered when reading ASCII files:

Empty file error. Make sure that you spelled the filename correctly and that it is in the current directory. If it is not, either copy it to the current directory or specify the complete filename (path plus full filename).

Long INPUT statement. If your INPUT statement is too long for one line, type a comma at the end of the first line and press Enter. Continue typing the statement on the next line. Do this for as many lines as you need.

Data lost or in the wrong columns. If SYSTAT places data incorrectly or data are lost when you read them, make sure that you correctly specify missing values in your data file. If you are using free-format input, enter missing numeric values as periods (.) and missing character values as blanks enclosed in quotation marks (" "). If you are using fixed-format input, you can leave missing values as blank spaces. If you are using free-format input to read a file that does not have the same number of values in every record, add the backslash (\) at the end of your INPUT statement.

Unexpected data error. Make sure that the ASCII file includes no field headings or variable labels, and make sure that variable types match data types. Do not put character data under a numeric variable or vice versa. Make sure that you correctly specify missing values. If you are using free-format input, enter missing numeric values as periods (.) and missing character values as blanks enclosed in quotation marks (" "). If you are using free-format input to read a file that does not have the same number of values in every record, add the backslash (\) to the end of your INPUT statement. If you are using fixed-format input, make sure that you specify the variable types in the format section correctly.

Non-ASCII character warning. Check for nonprinting characters in your file. Such characters include control characters, tab markers, margin and page-break indicators, and so on.

Nonmatching number of variables error. The number of variables defined in the format of your INPUT statement does not match the number of variables named in the variable list.

Input past end of record error. The format of your INPUT statement tells SYSTAT to read your ASCII file records further than allowed. This message should rarely occur.

Example 12

Unpacking Records

This example shows how to reorganize several repeated measures on a single record as one measure per record and how to generate a sequence number for each value.

Begin with the *TRIAL.SYZ* data file that has two records:

```
10 20 30 40 50 Male
11 21 31 41 51 Female
```

The following examples generate a file with 10 cases. First, you create a temporary ASCII text file with 10 cases, each containing one data value plus its sequence number and label:

```
USE TRIAL
PUSH CLASSIC
CLASSIC ON
OUTPUT TEMP
FOR I=1 to 5
    PRINT X(I), I, SEX$
NEXT
OUTPUT *
POP CLASSIC
```

SYSTAT's FOR... NEXT statement lists the five data values, and the values of *SEX\$* and *I* (the index) as sequence numbers. The OUTPUT * command closes the text file. You read the data from the text file *TEMP* into the SYSTAT file *NEWTRIAL* as follows:

```
GET TEMP
INPUT X, I, SEX$
ESAVE NEWTRIAL
```

The following are the values in the SYSTAT file *NEWTRIAL*:

	X	I	SEX\$
1	10.000	1.000	Male
2	20.000	2.000	Male
3	30.000	3.000	Male
4	40.000	4.000	Male
5	50.000	5.000	Male
6	11.000	1.000	Female
7	21.000	2.000	Female
8	31.000	3.000	Female
9	41.000	4.000	Female
10	51.000	5.000	Female

Export Data to an ASCII Text File

You may need to write data in SYSTAT files as text when you want to use them in a word processor or in another program. You can use the PUT command, the PRINT command, or the LIST command to store data in a text file.

If you want a command conversion procedure, use PUT:

```
USE filename
PUT output filename
```

where USE identifies the SYSTAT input file and PUT assigns a filename for the output ASCII file. PUT saves your data in a text file using 12 columns for each value, with values separated by commas. Character values (strings) are enclosed in double quotation marks (" "). When there are many variables, each case wraps such that no line of the ASCII file exceeds 132 columns in length. Use FORMAT n to control the number of decimal places appearing for numeric values. Variable names are not included in the ASCII file. SYSTAT automatically adds a .DAT.

Example 13

Converting a Data File to a Text File

The following commands convert the SYSTAT file *MINIWRLD* to a text file:

```
USE MINIWRLD
PUT TEXTFILE
```

You can use a word processor to view *TEXTFILE.DAT*. The first case in the file appears as follows:

```
"France      ", 56.358, 73.000, 14.000, 6.000,
14542.657, 648.069, 728.249, 432.780,
"Democracy  ", 73.000, 82.000, 99.000, "Europe", 1.556
```

While a hard return usually appears at the end of the text, the text for the first line "wraps around" here for display purposes. Notice that PUT adds commas as data separators and encloses character values in quotation marks. This does not happen if you use PRINT.

Example 14***Saving Data as Text without Quotes: PRINT***

If you do not want commas as delimiters and quotation marks around character data, use the PRINT command to send the output to a file.

```
USE MINIWRDL
IDVAR COUNTRY$
PUSH CLASSIC
CLASSIC ON
OUTPUT SUBSET2
PRINT COUNTRY$, LITERACY, URBAN, BIRTH_RT, GOV$
OUTPUT *
POP CLASSIC
```

The OUTPUT * command closes the text file. The resulting file, *SUBSET2.DAT*, looks like this:

France	99.000	73.000	14.000	Democracy
Greece	95.000	65.000	11.000	Democracy
Switzerland	99.000	58.000	12.000	Democracy
Spain	97.000	91.000	11.000	Democracy
UK	99.000	76.000	14.000	Democracy
Hungary	99.000	54.000	12.000	OneParty
Iraq	55.000	68.000	46.000	OneParty
Pakistan	26.000	28.000	43.000	Military
Ethiopia	55.200	14.000	45.000	Military
Afghanistan	12.000	16.000	44.000	OneParty
Brazil	76.000	68.000	26.000	Democracy
Canada	99.000	76.000	14.000	Democracy
Venezuela	85.600	76.000	28.000	Democracy
ElSalvador	65.000	39.000	34.000	Military

Select, List, and Save Cases

Use IF...THEN DELETE statements to select a subset of cases and store them in a SYSTAT data file, or use PUT or PRINT to save them in a text file. (You can also save selected cases using the SELECT command with EXTRACT.)

```
USE MYFILE
IF condition THEN DELETE
ESAVE subfile
```

or

```
USE MYFILE
IF condition THEN DELETE
PUT subtext
```


Example 15

Deleting Selected Cases

The following commands drop certain cases and list the variable *COUNTRY\$* to show the cases SYSTAT retained:

```
USE MINIWRDL
IF Case <=9 THEN DELETE
LIST COUNTRY$
```

SYSTAT responds:

Case	COUNTRY\$
1	Afghanistan
2	Brazil
3	Canada
4	Venezuela
5	ElSalvador

Example 16

Saving Selected Cases to a New File

The following commands save only those cases in *MINIWRDL* where *GDP_CAP* is greater than 10,000. The results are stored in *NEWFILE*.

The input is:

```
USE MINIWRDL
IDVAR COUNTRY$
IF GDP_CAP < 10000 THEN DELETE
ESAVE NEWFILE
LIST
```

The output follows:

Case ID	COUNTRY\$ EDUC LITERACY	POP_1990 HEALTH GROUP\$	URBAN MIL B_TO_D	BIRTH_RT GOV\$	BABYMORT LIFEEXPM	GDP_CAP LIFEEXPF
France	France	56.358	73.000	14.000	6.000	14542.657
	648.069	728.249	432.780	Democracy	73.000	82.000
	99.000	Europe	1.556			
Switzerland	Switzerland	6.742	58.000	12.000	5.000	17723.499
	853.538	1209.077	330.769	Democracy	75.000	83.000

	99.000	Europe	1.333			
Spain	Spain	39.269	91.000	11.000	6.000	10153.121
	166.675	224.923	117.964	Democracy	75.000	82.000
	97.000	Europe	1.375			
UK	UK	57.366	76.000	14.000	7.000	14259.401
	474.488	479.064	447.473	Democracy	73.000	79.000
	99.000	Europe	1.273			
Canada	Canada	26.538	76.000	14.000	7.000	19353.214
	1052.813	936.211	316.797	Democracy	74.000	81.000
	99.000	NewWorld	2.000			

Instead of using an IF...THEN DELETE statement to delete cases that are less than 10,000, you could have used SELECT:

```
SELECT GDP_CAP >= 10000
EXTRACT NEWFILE
```

Example 17

Omitting Variable Names and Case Numbers

The PRINT command is similar to LIST, except that case numbers and variable names are not printed. Also, PRINT can be used with IF THEN... DELETE, allowing you to print the first *k* cases. Type:

```
USE MINIWRDL
IF case>3 THEN DELETE
PRINT COUNTRY$, LITERACY, URBAN, BIRTH_RT
```

SYSTAT responds:

France	99.000	73.000	14.000
Greece	95.000	65.000	11.000
Switzerland	99.000	58.000	12.000

Case numbers and variables are not printed. SYSTAT prints numeric and character values in 12-column, right-justified fields. Blanks pad the left side of numeric fields. With PAGE set to WIDE, you can display up to 10 variables per line. If you use FORMAT to set the field width to fewer than 12 characters per field, you can display more than 10 variables per line.

Example 18

Using PRINT to Save Selected Cases

You can select specific cases by using the IF...THEN PRINT command. The example uses the OUTPUT command to store the values of *GROUP*\$, *COUNTRY*\$, *LITERACY*,

BIRTH_RT, and *LIFEEXPF* (for Islamic countries only) in the text file *TEXTFILE.DAT*:

```
USE OURWORLD
PUSH CLASSIC
CLASSIC ON
OUTPUT TEXTFILE
IF group$="Islamic" THEN PRINT COUNTRY$, LITERACY,,
                                     BIRTH_RT, LIFEEXPF
OUTPUT *
POP CLASSIC
```

The *OUTPUT ** command closes the text file. The ASCII file *TEXTFILE.DAT* looks like this:

Gambia	25.100	48.000	50.000
Iraq	55.000	46.000	68.000
Pakistan	26.000	43.000	57.000
Bangladesh	29.000	42.000	53.000
Ethiopia	55.200	45.000	52.000
Guinea	20.000	47.000	44.000
Malaysia	65.000	29.000	71.000
Senegal	28.100	44.000	56.000
Mali	18.000	51.000	47.000
Libya	50.000	37.000	70.000
Somalia	11.600	47.000	54.000
Afghanistan	12.000	44.000	46.000
Sudan	31.000	44.000	55.000
Turkey	70.000	29.000	67.000
Algeria	52.000	37.000	64.000
Yemen	15.000	52.000	49.000

Example 19

Printing the First Three Cases

If you want to see only some of the cases, you can specify the number of cases by using the *IF* condition *THEN DELETE* option. The following example sends the first three cases of *MINIWRLD* to the *NEWFILE.DAT* file:

```
USE MINIWRLD
PUSH CLASSIC
CLASSIC ON
OUTPUT NEWFILE
IF case>3 THEN DELETE
PRINT COUNTRY$, LITERACY, URBAN, BIRTH_RT
OUTPUT *
POP CLASSIC
```


Programming Examples

The examples in this section show more applications of SYSTAT transformation commands, including statistical calculations and data management procedures. There are simpler ways to accomplish many of these tasks, but the programs in this chapter are selected to illustrate the full range of SYSTAT capabilities through programming.

Example 20 Calendar for the Year 2007-2008

The commands to generate a calendar for the year 2007-2008 are shown below, and defined throughout the sections that follow.

```
PUSH CLASSIC
CLASSIC ON
FORMAT 0
NEW
ARRAY DAY(7)$
ARRAY DATE(7)$
FOR YEAR=2007 to 2008
  FOR MONTH=1 TO 12
    BASE=DOC(YEAR,MONTH,1)-1
    LAST=BASE+31
    FIRST=INT((BASE+1)/7)*7
    MONTH$=STR$(BASE+1,'MMMMMMMM')
    PRINT '+-----',MONTH$,YEAR,'-----+'
    FOR i=1 TO 7
      DAY(i)=Mid$(DOW$(i-1),1,3)
    NEXT
    PRINT
    DAY(1),' ',DAY(2),' ',DAY(3),' ',DAY(4),' ',DAY(5),' ',DAY(6),' ',DAY(7)
    FOR I= FIRST TO LAST STEP 7
      FOR L=I TO I+6
        TEMP=L+1-I
        IF L-BASE<10 THEN DATE(TEMP)= CAT$ (' ', STR$ (L-BASE)),
          ELSE DATE(TEMP)= STR$ (L-BASE)
          IF (DAT(L,'M')<>MONTH) THEN DATE(TEMP) = ' .'
      NEXT
    PRINT DATE(1),' ', DATE(2),' ', DATE(3),' ', DATE(4),' ',
    DATE(5),' ', DATE(6),' ',DATE(7)
    NEXT
    PRINT '+-----', '-----+'
    PRINT ' '
    PRINT ' '
    PRINT ' '
  NEXT
  CLEAR ARRAYS=DAY, DATE
POP CLASSIC
```


The calendar follows:

+----- January 2007 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	.	.	.

+----- February 2007 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	.	.	.	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	.	.	.

+----- March 2007 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	.	.	.	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

+----- April 2007 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30

+----- May 2007 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	.	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26

27	28	29	30	31	.	.
+----- June 2007 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
.
+----- July 2007 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31
+----- August 2007 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	.	.	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	.
+----- September 2007 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30
+----- October 2007 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20

21	22	23	24	25	26	27
28	29	30	31	.	.	.
+----- November 2007 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	.	.	.	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	.
+----- December 2007 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31
+----- January 2008 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	.	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	.	.
+----- February 2008 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	.
+----- March 2008 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	1
2	3	4	5	6	7	8

9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31

+----- April 2008 -----+

Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	.	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	.	.	.

+----- May 2008 -----+

Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	.	.	.	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

+----- June 2008 -----+

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30

+----- July 2008 -----+

Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	.	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	.	.

+----- August 2008 -----+

Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	1	2
3	4	5	6	7	8	9

10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31
+----- September 2008 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30
+----- October 2008 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	.	.	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	.
+----- November 2008 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30
+----- December 2008 -----+						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
.	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	.	.	.

Program Setup in BASIC

You will typically use the following setup:

```
USE filename
Program statements
.
.
.
ESAVE NEWFILE
```

SYSTAT executes the commands for each case in your data file before moving to the next case.

To preserve your work, you must save the transformed values to a new file. SYSTAT does not add these values to the original file. If you do not save your file, SYSTAT stores the results in a temporary data file.

When you enter a SYSTAT transformation command statement, SYSTAT reads it, checks for syntax errors, and then executes it. If SYSTAT finds an error, it tells you and lets you enter a new statement and continue programming. The statement with an error is forgotten.

Statements and Expressions

SYSTAT transformation commands make use of operators and functions in expressions and statements. When the data are incomplete (that is, the values are missing), the results of transformation command statements are altered. Arithmetic involving missing values propagates missing values. SYSTAT lists missing values first in numeric and alphabetical sorts and evaluates them as missing in relational comparisons.

IF...THEN Statement

With the IF...THEN statement, you can execute actions conditionally. The syntax for an IF...THEN statement is:

```
IF condition THEN action
```


The action that follows THEN can be any legal statement including LET, BEGINBLOCK...ENDBLOCK, DELETE, and another IF...THEN. For example, legal IF...THEN statements are:

```
IF condition THEN LET variable = expression
IF condition THEN PRINT variable and/or string
IF condition THEN DELETE
IF condition THEN IF condition THEN...
IF condition THEN BEGINBLOCK...ENDBLOCK
```

The action that follows THEN cannot be DIM, NEXT, GET, INPUT, TYPE, DIAGONAL, NEW, ARRAY, or SAVE, as these are not programming language statements.

You can execute more than one conditional transformation per cycle. If you are testing consecutive IF...THEN conditions on the same variable or variables, you should use IF...THEN...ELSE. Every IF...THEN block should end with the ENDIF statement. For example, in the above specified list of IF THEN statements, the last two should end with ENDIF.

ELSE Statement

An IF...THEN statement only tests cases for one condition (for example, $CARDIO > 325$, or $CARDIO > 325$ and $CANCER > 100$). If the case meets the condition, SYSTAT executes the transformation. If the case does not meet the condition, SYSTAT does not execute the transformation.

You may want, however, to execute many conditional transformations at once. If you are testing consecutive related conditions on the same variable, SYSTAT provides an ELSE statement to accompany IF...THEN. In its simplest form, IF...THEN and ELSE take the format:

```
IF condition THEN action1,
ELSE action2
```

SYSTAT executes *action2* only when the preceding IF condition evaluates to false. Another IF...THEN statement can follow ELSE, enabling you to string together a number of related conditional transformations:

```
IF condition THEN LET variable = expression,
ELSE IF condition THEN LET variable = expression,
ELSE IF condition THEN LET variable = expression,
ELSE LET variable = expression
ENDIF
```


In this case, SYSTAT executes the statement following ELSE only when all preceding IF conditions are false. When a preceding condition is true, SYSTAT ignores subsequent ELSE statements.

Example 21

IF...THEN...ELSE Statement

The following examples compare two sets of transformation commands. The first uses only IF...THEN statements to assign values to a new variable called *RATE\$* based on values for *CARDIO*:

```
USE USSTATES
  IF CARDIO < 275 THEN LET RATE$ = 'LOW'
  IF CARDIO >= 275 AND CARDIO < 325,
    THEN LET RATE$ = 'AVERAGE'
  IF CARDIO >= 325 THEN LET RATE$ = 'HIGH'
  IF CARDIO = . THEN LET RATE$ = 'MISSING'
ESAVE NEWDATA
```

Using IF...THEN and ELSE makes the commands simpler:

```
USE USSTATES
  IF CARDIO < 275 THEN LET RATE$ = 'LOW',
  ELSE IF CARDIO < 325 THEN LET RATE$ = 'AVERAGE',
  ELSE IF CARDIO >= 325 THEN LET RATE$ = 'HIGH',
  ELSE LET RATE$ = 'MISSING'
ENDIF
ESAVE NEWDATA
```

SYSTAT executes these commands once for each case. The order of the IF and ELSE statements is important. The ELSE depends on the truth of the IF conditions before it. SYSTAT executes an ELSE statement only if all preceding conditions are false.

After running these commands, check the values for the first 10 cases for *CARDIO* and *RATE\$*:

```
USE NEWDATA
LIST STATE$, CARDIO, RATE$ / N=10
```


The output follows:

Case	STATES	CARDIO	RATES
1	ME	330.400	HIGH
2	NH	278.300	AVERAGE
3	VT	289.400	AVERAGE
4	MA	343.900	HIGH
5	RI	353.600	HIGH
6	CT	318.800	AVERAGE
7	NY	376.200	HIGH
8	NJ	356.100	HIGH
9	PA	400.600	HIGH
10	OH	344.600	HIGH

FOR...NEXT Statement

The syntax for a FOR...NEXT statement is:

```

FOR index = n1 TO n2 STEP n3
    statement
    statement
    .
    .
    .
NEXT

```

Note that STEP is optional. The following are some examples:

```

FOR i = 1 TO 10
    PRINT i
NEXT

FOR j = 1 TO 10
    LET x(j) = LOG(x(j))
NEXT

```

Instead of the second FOR...NEXT statement, you could have stated:

```

LET (X(1)...X(10))=LOG(@)

```

FOR...NEXT loops are executed for each case (or the number of times specified in a SELECT statement). In the first example, the PRINT statement is executed 10 times for each case. The second example computes the natural log for each of 10 subscripted variables.

FOR...NEXT loops are tested at the beginning to determine whether they should be executed for the current value of index. Some other languages can execute FOR...NEXT

loops once even when a condition is false. The following example will not print anything:

```
FOR i = 6 TO 3 STEP 1
  PRINT i
NEXT
```

Nesting. You can use nested FOR...NEXT loops as follows:

```
FOR
  FOR
    FOR
      .
      .
      .
    NEXT
  NEXT
NEXT
```

Always match every FOR with a NEXT.

WHILE...ENDWHILE Loop

The WHILE ...ENDWHILE loop works similar to the FOR...NEXT loop. It goes with a condition and a series of commands in its loop. The commands in the loop are executed as long as the condition is satisfied. Only temporary variables are allowed to use in the condition. But inside the loop you can use data variables.

The syntax for a WHILE ...ENDWHILE statement is:

```
WHILE (condition)
  commandlist
ENDWHILE
```

Note: If one needs a multiple PRINT or WHILE loop in an IF...THEN statement it must start on a new line as shown below:

```
IF condition THEN
  PRINT varlist
ELSE LET | DELETE
  PRINT varlist
ENDIF
```

```
IF condition THEN
  WHILE (condition)
    ...
  ENDWHILE
ENDIF
```


Example 22**IF statement with BEGINBLOCK...ENDBLOCK**

The mean value for the variable *PULMONAR* from the data set *USSTATES* is approximately 34.8 with a standard deviation of 7.7. Suppose you want to set *RATE\$* to *HIGH* and *RATE* to 1 if *PULMONAR* is more than one standard deviation above average.

```
USE USSTATES
IF PULMONAR > 34.8 THEN BEGINBLOCK
  LET RATE$ = 'HIGH'
  LET RATE = 1
ENDBLOCK
ENDIF
LIST PULMONAR, RATE$, RATE
```

Here are the first 10 cases:

Case	PULMONAR	RATE\$	RATE
1	46.100	HIGH	1.000
2	33.500		.
3	45.200	HIGH	1.000
4	34.400		.
5	34.300		.
6	29.300		.
7	31.000		.
8	29.300		.
9	36.000	HIGH	1.000
10	37.500	HIGH	1.000

For the cases that meet the condition *PULMONAR* > 34.8, SYSTAT executes the two transformations between *BEGINBLOCK* and *ENDBLOCK*. For those cases that do not meet the condition, SYSTAT assigns missing values to *RATE\$* and *RATE*. Missing values for string variables appear as blanks.

Example 23**BEGINBLOCK...ENDBLOCK with ELSE Statement**

This example shows the IF...THEN...ELSE format with the *BEGINBLOCK*...*ENDBLOCK* statement and the following assignments:

Where PULMONAR is	Let RATE\$ =	Let RATE =
< 20.8	LOW	1
>= 20.8 and < 34.8	MID	2
>= 34.8	HIGH	3

The input is:

```

USE USSTATES
  IF PULMONAR < 20.8 THEN BEGINBLOCK
    LET RATE$ = 'LOW'
    LET RATE = 1
  ENDBLOCK
  ELSE IF PULMONAR < 34.8 THEN BEGINBLOCK
    LET RATE$ = 'MID'
    LET RATE = 2
  ENDBLOCK
  ELSE IF PULMONAR >= 34.8 THEN BEGINBLOCK
    LET RATE$ = 'HIGH'
    LET RATE = 3
  ENDBLOCK
ENDIF
LIST PULMONAR, RATE$, RATE

```

Here are the first 10 cases:

Case	PULMONAR	RATE\$	RATE
1	46.100	HIGH	3.000
2	33.500	MID	2.000
3	45.200	HIGH	3.000
4	34.400	MID	2.000
5	34.300	MID	2.000
6	29.300	MID	2.000
7	31.000	MID	2.000
8	29.300	MID	2.000
9	36.000	HIGH	3.000
10	37.500	HIGH	3.000

If, for a case, *PULMONAR* is less than 20.8, SYSTAT executes the associated *BEGINBLOCK...ENDBLOCK* statements, setting the values of *RATE\$* to *LOW* and *RATE* to 1. It does not execute the subsequent *ELSE* statements but moves on to the next case.

If *PULMONAR* is greater than or equal to 20.8 but less than 34.8, SYSTAT executes the first *ELSE* statement. SYSTAT sets *RATE\$* to *MID* and *RATE* to 2 and does not execute the second *ELSE* statement.

If *PULMONAR* is greater than 34.8, SYSTAT executes the last *ELSE* statement and sets *RATE\$* to *HIGH* and *RATE* to 3.

FOR...NEXT Loops with Subscripted Variables

You can use *FOR...NEXT* to define program loops that assign incremental values to an index variable. You can use such a loop to transform a set of subscripted variables. SYSTAT executes the statements between the *FOR* and the *NEXT* statements for each

successive value of the index variable you specify. The index variable begins with the initial value you assign, does the transformations, and then increases by one. The cycle repeats until the index variable reaches the limit specified with TO. For example:

```
FOR i = 1 TO 5
```

```
FOR j = 2 TO 20 STEP 2
```

The STEP option adjusts the size of the increment. If you enter the following input, SYSTAT increments n by two each time.

```
FOR n = 1 TO 10 STEP 2
```

The values of n are listed as 1, 3, 5, 7, and 9 consecutively. With this command, SYSTAT runs through the loop only five times.

If you want to execute a set of commands on a certain number of cases, use the IF ... THEN DELETE or SELECT command rather than the FOR...NEXT construct. Remember, every set of commands is executed once for each case; therefore, if you use FOR...NEXT, the loop runs for every case. This is possible only when the FOR...NEXT loop contains any data variable inside the loop.

Temporary Subscripts Using the ARRAY Statement. If your variable names are not already subscripted, you can use the ARRAY statement to assign subscripts temporarily for the purpose of doing transformations inside a FOR...NEXT loop.

Example 24 ***Logging Ten Variables***

Suppose you have a file containing the variables $X(1...10)$ and you want to calculate the natural log of each. You could either enter 10 separate LET commands or use the FOR...NEXT looping construct to do this. The input is as follows:

```
FOR n = 1 TO 10
  LET x(n) = LOG(x(n))
NEXT
```


SYSTAT runs through the loop 10 times, increasing the value of n by one each time. Thus, the value of n increases successively: 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10. This FOR...NEXT statement is the same as:

```
LET x(1) = LOG(x(1))
LET x(2) = LOG(x(2))
LET x(3) = LOG(x(3))
LET x(4) = LOG(x(4))
LET x(5) = LOG(x(5))
LET x(6) = LOG(x(6))
LET x(7) = LOG(x(7))
LET x(8) = LOG(x(8))
LET x(9) = LOG(x(9))
LET x(10) = LOG(x(10))
```

Alternatively, you could use the @ shortcut and specify:

```
LET (x(1) .. x(10)) = LOG(@)
```

If you do not want to change the values in x by replacing them with their logs, you can assign the transformed values to a new variable y . (Use the DIM statement to create a new subscripted variable.)

The STEP option adjusts the size of the increment. The following increases n by two each time to values 1, 3, 5, 7, and 9 successively. SYSTAT runs through the loop only five times.

```
FOR n = 1 TO 10 STEP 2
```

This is the same as:

```
LET x(1) = LOG(x(1))
LET x(3) = LOG(x(3))
LET x(5) = LOG(x(5))
LET x(7) = LOG(x(7))
LET x(9) = LOG(x(9))
```

Dimensioning Space for New Variables

To add new subscripted variables to a file, use the DIM statement. A DIM statement reserves space for new subscripted variables. String variables can be used. For example, the following DIM statement creates new variables $X(1)$, $X(2)$, $X(3)$, $X(4)$, and $X(5)$.

```
DIM x(5)
```


You can use subscripted variables defined with DIM in transformations. Suppose you have variables $X(1...10)$. The following program creates new variables $Y(1...10)$ whose values are the natural logarithms of the corresponding values in $X(1...10)$.

```
DIM Y(10)
FOR n = 1 TO 10
  LET Y(n) = LOG(X(n))
NEXT
```

Without the DIM statement, SYSTAT would not understand the y variable subscript in the LET statement and would respond with an error message. You cannot redimension existing or previously defined arrays of subscripted variables.

Example 25 **Computing Means of Subscripted Variables**

The following commands compute the average of the variables $X(1)$ through $X(10)$ for each case. The statement

```
IF X(i) <> .
```

checks for missing data. (Of course, you can calculate the mean more easily with the multivariable function AVG.)

```
USE MYFILE
DIM X(10)
LET SUMX = 0
LET N = 0
FOR I = 1 TO 10
  IF X(I) <> . THEN BEGINBLOCK
    LET SUMX = SUMX + X(I)
    LET N = N + 1
  ENDBLOCK
NEXT
IF N <> 0 THEN LET MEAN = SUMX/N
ELSE LET MEAN = .
ENDIF
ESAVE NEWDATA
```

The commands beginning with LET and ending with ELSE are executed once for each case. At the start of each case, LET SUMX=0 and LET N=0 set the variables *SUMX* and *n* to 0. *SUMX* sums the nonmissing values across each case, and *n* counts the nonmissing values for each case.

The FOR...NEXT loop runs the variables $X(1...10)$ through two conditional transformations. In the first, if $X(1)$ is not missing, its value is added to $SUMX$. In the second, again if $X(1)$ is not missing, the count variable n is increased by one.

Upon completion of the FOR...NEXT loop, another conditional transformation tests whether n is not equal to 0. If n is not equal to 0, then the calculation $MEAN = SUMX/N$ is executed. If n equals 0 (because the values of $X(1...10)$ for the current case are all missing), dividing by n would cause an error. Therefore, SYSTAT executes the ensuing ELSE statement and sets $MEAN$ to missing (.).

Example 26

Computing Means of Unsubscripted Variables: ARRAY

To average variables that are not subscripted, use the ARRAY command to alias the variables with a subscripted variable and then use the same logic employed for subscripted variables.

You can have only one ARRAY command per set of commands. The ARRAY command can be used only in conjunction with a FOR...NEXT loop.

The example below averages the values of the average expenditures for health, education, and the military (*HEALTH*, *EDUC*, and *MIL*) using the *OURWORLD* data file and tests for missing values.

```
USE OURWORLD
ARRAY DOLLARS / HEALTH EDUC MIL
LET SUMMONEY = 0
LET N = 0
FOR I = 1 TO 3
  IF DOLLARS(I) <> . THEN BEGINBLOCK
    LET SUMMONEY = SUMMONEY + DOLLARS(I)
    LET N = N + 1
  ENDBLOCK
ENDIF
NEXT
IF N <> 0 THEN LET MEAN = SUMMONEY/N
ELSE LET MEAN = .
ESAVE NEWDATA
```

SYSTAT treats each variable specified in the ARRAY statement as an element in a subscripted variable named *DOLLARS*.

```
HEALTH = DOLLARS(1)
EDUC = DOLLARS(2)
MIL = DOLLARS(3)
```


Subgroup Processing: BOG, EOG, BOF, EOF

You can specify SYSTAT commands that operate on subgroups of cases in a file. To do this, you must first identify variables in your file which define subgroups. SYSTAT has four special variables available for processing subgroups:

- BOF has value 1 if beginning-of-file, else it is 0.
- EOF has value 1 if end-of-file, else it is 0.
- BOG has value 1 if beginning-of-BY group, else it is 0.
- EOG has value 1 if end-of-BY group, else it is 0.

Before using BOG or EOG, you must use a BY statement to identify the variables that define subgroups in your data. You can name up to 10 variables. To clear a previous BY command, type BY with no arguments.

Here, the BY command behaves slightly differently than in other procedures. Rather than causing subsequent commands to be executed on subgroups, it specifies which variable or variables are used for BOG and EOG.

You can use BOG, EOG, BOF, and EOF within conditional expressions in IF...THEN statements. For example, the statement

```
IF BOG THEN LET total =x
```

causes SYSTAT to execute the statement every time it encounters a new value in a BY variable. This is because the value of BOG is 1 ("true") for every case that begins a new group (and 0 otherwise).

Example 27

EOF: Printing the Last Case in a File

The following procedure prints the value of *CARDIO* for the last case in the *USDATA* file. EOF is 0 for every case but the last, where its value is 1.

```
USE USDATA
IF EOF THEN PRINT,
    "The CARDIO value for the last case is",CARDIO
```

To print all but the last case, set the condition to one of the following:

```
IF EOF=0 THEN...
IF NOT EOF THEN...,
```


Example 28

Computing Totals within Groups

Suppose you want to sum values of the variable *x* within *GROUP1* and within *GROUP2*. You start with a file called *OURFILE* and want to write a file called *OURFILE2*.

OURFILE		OURFILE2		
GROUP	X	GROUP	X	TOTAL
1	3	1	3	3
1	4	1	4	7
1	6	1	6	13
2	9	2	9	9
2	-1	2	-1	8
2	2	2	2	10

The commands to do this are:

```
USE OURFILE
LET TOTAL
BY GROUP
IF BOG THEN LET TOTAL=X,
ELSE LET TOTAL=LAG(TOTAL) + X
IF EOG THEN PRINT GROUP, TOTAL
ESAVE OURFILE2
```

BOG is true when a case is first in its group; otherwise, it is false.

Saving Only the Totals with Group Identifiers

If you want *OURFILE2* to include only the summary record for each group,

GROUP	TOTAL
1	13
2	10

insert:

```
IF NOT EOG THEN DELETE
```

after ESAVE and

DROP X

after ESAVE in the last setup. EOG means “end of group.”

Example 29 **Subgroup Means**

You can use a variable (*n*) that counts the number of cases within each group to compute the mean within each group:

$$\text{mean} = \text{TOTAL} / N$$

The data in the output file *OURFILE3* are:

GROUP	MEAN
1	4.333
2	3.333

The input follows:

```
USE OURFILE
LET TOTAL
LET N
BY GROUP
IF BOG THEN LET TOTAL=X,
ELSE LET TOTAL=LAG(TOTAL) + X
IF BOG THEN LET N=1,
ELSE LET N=LAG(N) + 1
ENDIF
IF EOG THEN LET MEAN=TOTAL/N
IF NOT EOG THEN DELETE
ENDIF
ESAVE OURFILE3
DROP X, TOTAL, N
```

PRINT Statement

The PRINT command prints the values of the variables you specify. You can also use PRINT to print character strings, which can be used in transformation commands.

Suppose you have a program that sums the values of a variable. Also suppose that instead of recording the answer (sum) somewhere in the worksheet, you just want the program to display its results. You can do this by issuing a PRINT statement:

```
PRINT "The sum of values in A is ", sum
```

PRINT in SYSTAT prints numeric values in 12-column, right-justified fields. Blanks pad the left of each field. Character values are left-justified. Character strings that you specify literally (that is, not those that are values of character variables you listed in the PRINT command) are not justified; they are printed exactly the way you specified them, but without the quotation marks.

Example 30

Generating Uniform Random Numbers with WHILE ... ENDWHILE loop

If you want to generate Uniform random numbers by using the WHILE ... ENDWHILE loop, then the command script is as follows:

```
NEW
REPEAT 20
RSEED 12345
DIM X (5)
I = 1
WHILE (I<=5)
LET X(i) = Urn (0,1)
I = I+1
ENDWHILE
LIST/N = 10
```

SYSTAT adds five new subscripted variables to a file using the DIM statement. With initialization of I = 1, the WHILE ... ENDWHILE loop will generate five sets of Uniform random numbers with Random Seed equal to 12345. These generated subscripted variables can be viewed in the data file.

The output is:

Case	X(1)	X(2)	X(3)	X(4)	X(5)
1	0.813	0.218	0.097	0.003	0.957
2	0.129	0.582	0.044	0.663	0.518
3	0.255	0.781	0.254	0.519	0.459
4	0.776	0.500	0.675	0.308	0.421
5	0.027	0.189	0.604	0.150	0.114
6	0.027	0.189	0.604	0.419	0.262
7	0.595	0.852	0.216	0.083	0.155
8	0.666	0.490	0.762	0.727	0.621
9	0.786	0.065	0.925	0.339	0.319
10	0.674	0.942	0.933	0.011	0.545
11	0.983	0.726	0.120		

Example 31

IF...THEN DELETE: Using the first N Cases

For any set of SYSTAT commands, you can use the IF...THEN DELETE command to limit the computations to a certain number of cases—just as you can use SELECT Case <=10 to limit the action of commands like LIST to the first 10 cases.

You can use IF...THEN DELETE to test complex transformations. If you use IF...THEN DELETE, you can see whether the commands are correct or if you need to change them before running it on an entire file. For example:

```
USE MYFILE
IF Case>3 THEN DELETE
LET X = LOG(A)
LET Y = LOG(B)
PRINT X, Y
```

If you made a mistake while writing the program, you would find the error before running the program on the entire file. For files with several hundred cases, a brief trial run can save time. If you use SELECT <=3 in place of IF case>3 THEN DELETE, then after you correct any errors, type SELECT with no arguments to restore the counter.

Example 32

Random Subsamples

The examples below illustrate two methods of taking random samples without replacement from data files. The first extracts a percentage of cases from a file; the second, a specific number of cases.

To pick a random sample of approximately three-fourths of a file, type:

```
USE USSTATES
IF URN () > .75 THEN DELETE
```

To vary the sample size, change the 0.75 proportion to another number between 0 and 1.

Here is another method which keeps both the selected and deselected cases in the same file. The *WEIGHT* variable can be used with statistical procedures to select the random subsample for cross-validation.

```
USE USSTATES
IF URN () > .75 THEN LET WEIGHT = 0,
ELSE LET WEIGHT = 1
```


Example 33**Fitting Normal Distribution with FOR...NEXT Statement**

Suppose you want to generate five variables from a Normal distribution and test the goodness of fit. Then the input is as follows:

```

NEW
REPEAT 50
DIM X(5)
FOR i = 1 to 5
LET X(i) = Zrn(0,1)
NEXT

LET X
ARRAY A(5)$
A(1) = "X(1)"
A(2) = "X(2)"
A(3) = "X(3)"
A(4) = "X(4)"
A(5) = "X(5)"
FITDIST
FOR i = 1 to 5
LET X = X(i)
PRINT "The results of fitting Normal distribution to the,
variable",A(i), "are shown below with variable name as X:"
Continuous X / dist = Z
NEXT

```

Using the first set of commands, SYSTAT creates five Normal variables with mean zero and variance one. Then to test the goodness of fit, SYSTAT creates a temporary variable XX to execute the *FOR...NEXT* loop. Also `ARRAY A(5)$` will help us to create string variables X(1) to X(5). The results of fitting a Normal distribution to these data are shown below:

The results of fitting Normal distribution to the variable X(1) are shown below with variable name as X:

```

Variable Name : X
Distribution   : Normal

```

Estimated Parameter(s)

```

Location or Mean(mu) : -0.035
Scale or SD(sigma)   : 0.940

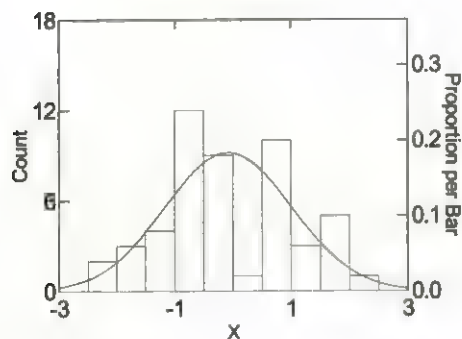
```

Estimation of Parameter(s): Maximum Likelihood Method

Test Results

Lower Limit	Upper Limit	Observed	Expected
.	-1.105	8	6.376
-1.105	-0.708	3	5.473
-0.708	-0.311	11	7.372
-0.311	0.086	3	8.330
0.086	0.482	8	7.898
0.482	0.879	6	6.282
0.879	.	11	8.269
		50	50.000

Chi-square Test Statistic : 7.643
 Degrees of Freedom : 4
 p-value : 0.106

Fitted Distribution

Kolmogorov-Smirnov Test Statistic : 0.086
 p-value : 0.448

Shapiro-Wilk Test Statistic : 0.980
 p-value : 0.552

The results of fitting Normal distribution to the variable X(2) are shown below with variable name as X:

Variable Name : X
 Distribution : Normal

Estimated Parameter(s)

Location or Mean(μ) : -0.020
 Scale or SD(σ) : 1.187

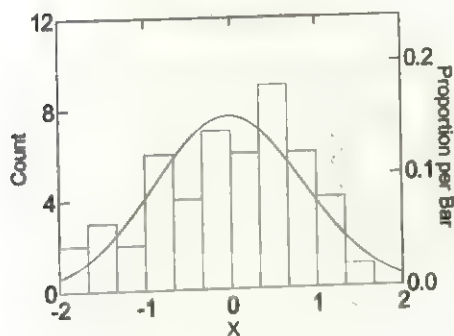
Estimation of Parameter(s): Maximum Likelihood Method

Test Results

Lower Limit	Upper Limit	Observed	Expected
.	-1.438	4	5.810
-1.438	-0.773	9	7.342
-0.773	-0.108	12	10.377
-0.108	0.557	10	10.804
0.557	1.222	8	8.285
1.222	.	7	7.382
		50	50.000

Chi-square Test Statistic : 1.281
 Degrees of Freedom : 3
 p-value : 0.734

Fitted Distribution



Kolmogorov-Smirnov Test Statistic : 0.090
 p-value : 0.380

Shapiro-Wilk Test Statistic : 0.963
 p-value : 0.118

The results of fitting Normal distribution to the variable X(3) are shown below with variable name as X:

Variable Name : X
 Distribution : Normal

Estimated Parameter(s)

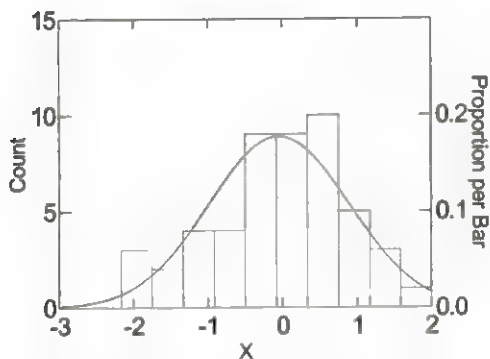
Location or Mean(μ) : 0.045
 Scale or SD(σ) : 0.907

Estimation of Parameter(s): Maximum Likelihood Method

Test Results

Lower Limit	Upper Limit	Observed	Expected
.	-1.044	5	5.756
-1.044	-0.587	6	6.396
-0.587	-0.131	10	9.015
-0.131	0.326	8	9.916
0.326	0.782	13	8.512
0.782	.	8	10.405

Chi-square Test Statistic : 3.524
 Degrees of Freedom : 3
 p-value : 0.318

Fitted Distribution

Kolmogorov-Smirnov Test Statistic : 0.072
 p-value : 0.745

Shapiro-Wilk Test Statistic : 0.985
 p-value : 0.765

The results of fitting Normal distribution to the variable X(4) are shown below with variable name as X:

Variable Name : X
 Distribution : Normal

Estimated Parameter(s)

Location or Mean(μ) : 0.276
 Scale or SD(σ) : 0.940

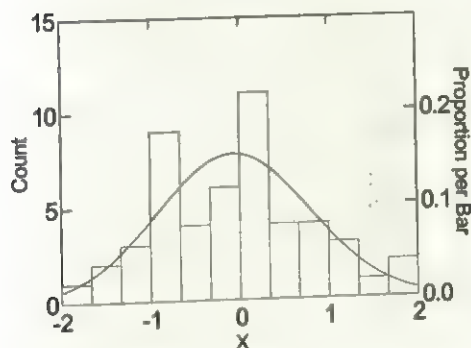
Estimation of Parameter(s): Maximum Likelihood Method

Test Results

Lower Limit	Upper Limit	Observed	Expected
.	-0.561	8	9.344
-0.561	-0.139	9	7.135
-0.139	0.283	11	8.667
0.283	0.704	8	8.639
0.704	1.126	2	7.066
1.126	.	12	9.148
		50	50.000

Chi-square Test Statistic : 5.877
 Degrees of Freedom : 3
 p-value : 0.118

Fitted Distribution



Kolmogorov-Smirnov Test Statistic : 0.102
 p-value : 0.208

Shapiro-Wilk Test Statistic : 0.978
 p-value : 0.455

The results of fitting Normal distribution to the variable X(5) are shown below with variable name as X:

Variable Name : X
 Distribution : Normal

Estimated Parameter(s)

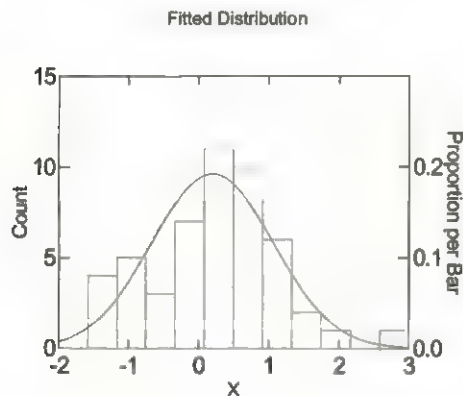
Location or Mean(μ) : 0.077
 Scale or SD(σ) : 1.016

Estimation of Parameter(s): Maximum Likelihood Method

Test Results

Lower Limit	Upper Limit	Observed	Expected
.	-0.723	12	10.772
-0.723	-0.209	8	8.683
-0.209	0.305	10	9.983
0.305	0.819	6	8.933
0.819	1.333	8	6.221
1.333	.	6	5.407
		50	50.000

Chi-square Test Statistic : 1.730
 Degrees of Freedom : 3
 p-value : 0.630



Kolmogorov-Smirnov Test Statistic : 0.079
 p-value : 0.583

Shapiro-Wilk Test Statistic : 0.961
 p-value : 0.094

Note: Use the RSEED option after REPEAT 50 to get the same set of generated normal variables.

Matrix

Laszlo Engelman

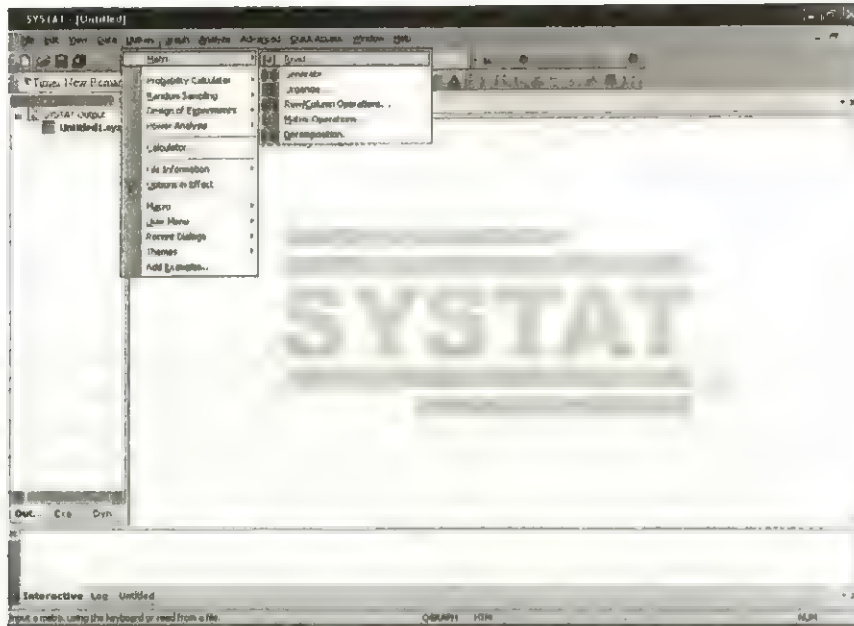
(revised by A.V.Kharshikar, Amit Saxena, S.R.Kulkarni, Nandita Ingawale, and Santosh Ranjan)

Most of the algorithms for classical analyses provided by statistical software packages can be expressed using matrix algebra; thus, they can be computed using the Matrix feature. Students will find Matrix useful for gaining an understanding of matrix algebra and how statistical computations work. Researchers can prototype state-of-the-art procedures before they become available in statistical packages, and can execute complex data management tasks.

Useful material on matrix algebra may be found in Searle (1982), Schott (1997), Harville (1997), and Rao and Rao (1998). These books discuss the use of matrices in Statistics. Rao (1973), Seber and Lee (2003), Kutner et al. (2004), and Birkes and Dodge (1993) have useful chapters or an appendix dealing with matrix algebra. Matrix in SYSTAT offers functions that allow you to use matrix algebra to perform statistical analyses and data management tasks. Matrix provides a flexible facility to manipulate matrices in addition to performing matrix algebra. Most of the standard matrix manipulations, operations, and computations can now be menu-driven. You have the choice, of course, to achieve them through commands as in earlier versions. Functions like QR, Cholesky, and singular value decomposition are available for many statistical analyses. Procedures like Kronecker product, raising a matrix to a power, and solving a consistent system of n equations in n variables are available. Reshaping a matrix, filling missing values, making a matrix symmetric, and many more facilities are provided to handle complex data management tasks.

Matrix in SYSTAT

The Matrix feature is organized as follows:

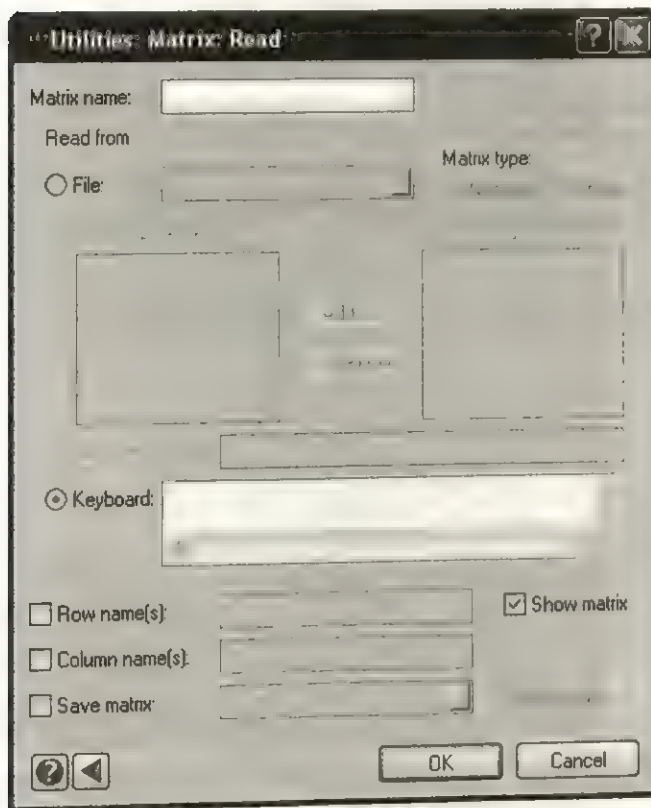


A matrix is referred to by its name. We refer the matrices in the memory as available matrices and the last handled matrix as the active matrix. Matrix allows as many matrices as the memory allows. A matrix may contain all columns with either numeric or character values. However, Matrix fails if a numeric operation is attempted on character data or if operands are mixed; for example, comparing a character datum to a numeric value.

Read Matrix Dialog Box

To open the Read Matrix dialog box, from the menus choose:

Utilities
Matrix
Read...



The following options can be specified:

Matrix name. Enter a name for the matrix that you want to create.

Read from. You need to choose one from the following options:

- **File.** Select a file that you want to read into a matrix.
 - **Matrix type.** Choose either numeric or the string type of the matrix.

- **Selected variable(s).** Variable(s) selected for representing the columns of the matrix.
- **Selected case(s).** Type the row numbers separated by commas or spaces, use double period (..) to specify a range of rows (e.g. 1..20).
- **Keyboard.** Enter the values separated by commas or spaces. Separate each row with a semicolon, if you type several rows on one line.

Row name(s). You can specify names for the rows of the matrix. Use commas or spaces to separate the names.

Column name(s). You can specify names for the columns of the matrix. Use commas or spaces to separate the names.

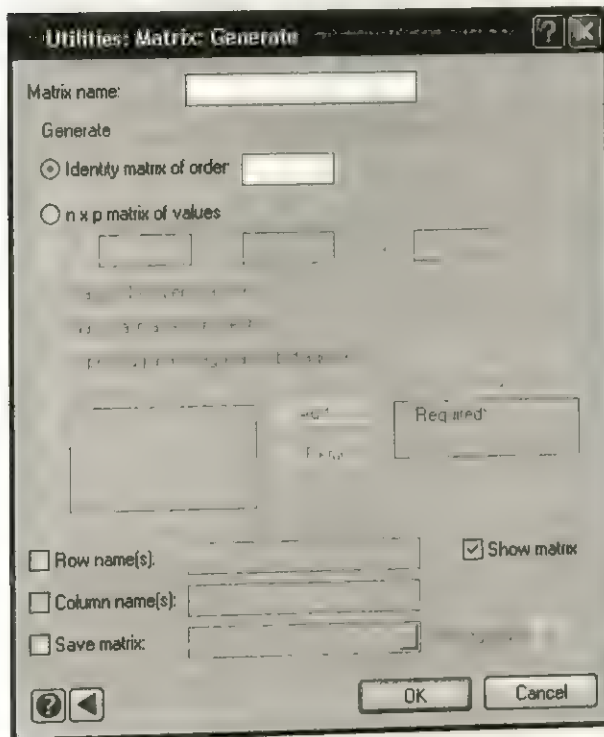
Save matrix. You can save the created matrix to a specified file. The following options are available: Rectangular, SSCP, Covariance, Correlation, Dissimilarity and Similarity.

Show matrix. Prints the created matrix to the current output device.

Generate Matrix Dialog Box

To open the Generate Matrix dialog box, from the menus choose:

Utilities
Matrix
Generate...



Matrix name. Enter a name for the matrix you want to generate.

Selected matrix/vector. Matrix selected for the desired operations.

Generate. Choose one from the following options:

- **Identity matrix of order.** Generates an identity matrix of a specified order.
- **$n \times p$ matrix of values.** Generates a rectangular matrix with n rows and p columns filled with a value that you specify.
- **Diagonal elements as row.** Extracts the diagonal elements of the selected square matrix.

- **Diagonal matrix from vector.** Generates a diagonal matrix with diagonal elements from the selected vector.
- **Super-diagonal; diagonal; sub-diagonal.** Extracts the super-diagonal, diagonal, and sub-diagonal of the selected square matrix.

Row name(s). You can specify names for the rows of the matrix. Use commas or spaces to separate the names.

Column name(s). You can specify names for the columns of the matrix. Use commas or spaces to separate the names.

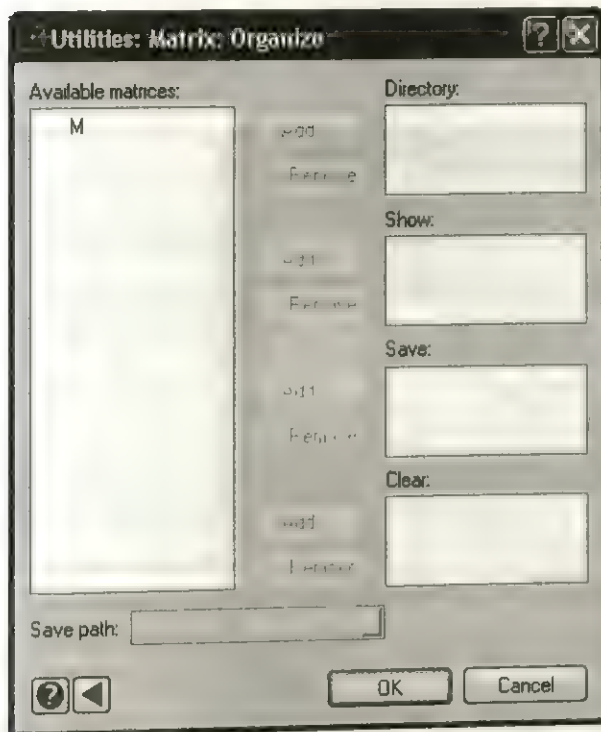
Save matrix. You can save the generated matrix to a specified file. The following options are available: Rectangular, SSCP, Covariance, Correlation, Dissimilarity and Similarity.

Show matrix. Prints the generated matrix to the current output device.

Organize Dialog Box

To open the Organize dialog box, from the menus choose:

Utilities
Matrix
Organize...



You can clear the desired matrices from memory, save the matrices, view matrices, and/or see the description of matrices.

Directory. Gives a description of the selected matrices.

Show. Prints the selected matrices to the current output device.

Save. You can save the selected matrices to a specified file.

Clear. Clears the selected matrices from memory.

Save path. Lets you choose the path for saving the matrices.

You can perform any of the above operations on the same matrix.

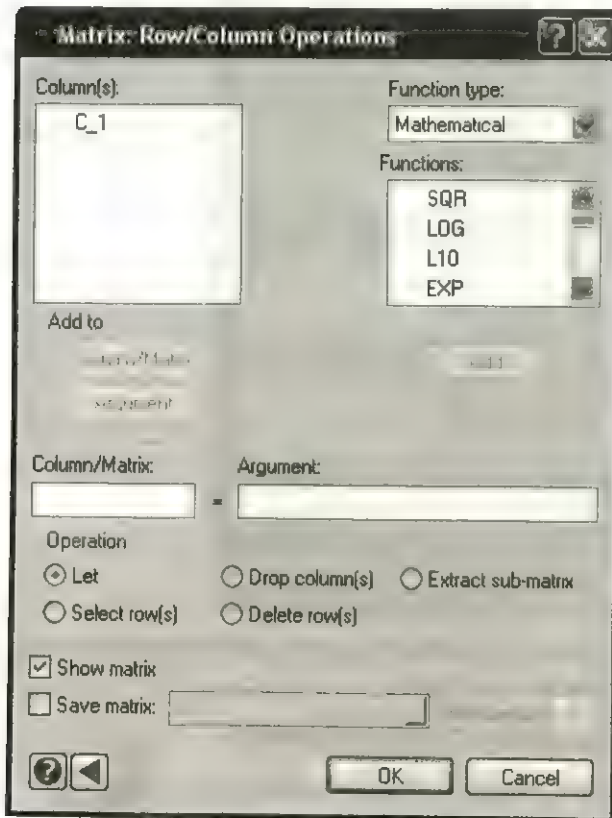
Row/Column Operations Dialog Box

To perform Row/Column operations on an active matrix, from the menus choose:

Utilities

Matrix

Row/Column Operations...



Column/Matrix and Argument. To specify the target column/matrix, select a column/matrix from the list and click the Column/Matrix button. If you want to create a new column (matrix), simply type it into the box. Select the function type, functions and column (matrix or matrices) to be used in the argument.

You can operate on existing matrices or create new ones. If the target column (matrix) already exists, it is replaced by the expression written in argument. If the matrix does not exist, then a new matrix with that name is created.

Operation. You need to choose one from the following options:

- **Let.** Creates a new column or alters an existing column in the active matrix. Note that LET works in the same manner here as LET does elsewhere in SYSTAT.

If the active matrix has a column named *NAME*, it is overwritten; if it does not, a new column is added. The expression can be any combination of column names, constants, mathematical functions, and operators. See Chapter 4: Data Transformations: “Let Dialog Box” on page 73 in *DATA* for more information about LET statements.

- **Drop column(s).** Deletes columns of the active matrix specified by variable names, column numbers, or a range of columns (for example, 10 .. 31).
- **Extract sub-matrix.** You can extract a sub-matrix from the active matrix. The expression for that is:

```
name = mat_name(ref to rows; ref to columns).
```

Reference to rows/columns can be a list or range of row/column numbers or a list of row/column names or a condition involving column names (for example AGE > 21). Use a double period (..) to specify a range of rows or columns.

- **Select row(s).** Retains only those rows of the active matrix that meet the specified condition. You can specify the condition using functions and column names.
- **Delete row(s).** Deletes rows of the active matrix specified by row names, row numbers, or a range of rows.

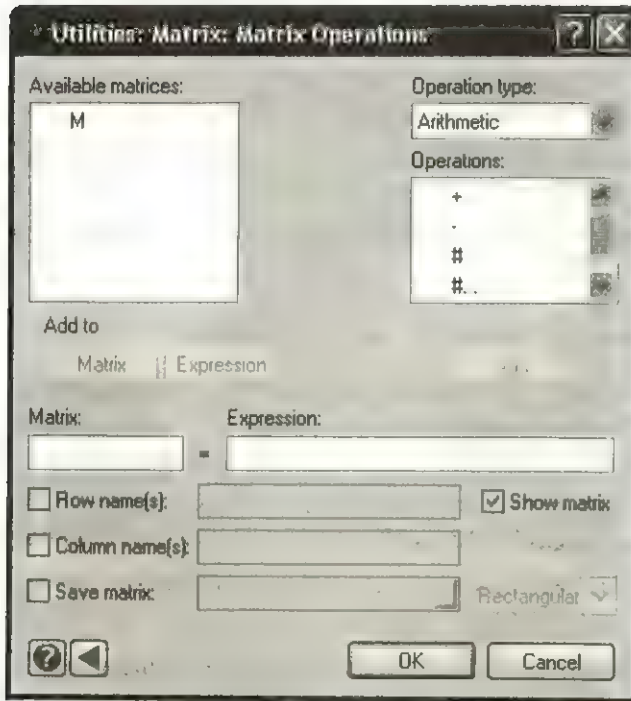
Show matrix. Prints the generated matrix to the current output device.

Save matrix. Saves the generated matrix to a SYSTAT data file in the specified path. The following options are available: Rectangular, SSCP, Covariance, Correlation, Dissimilarity and Similarity.

Matrix Operations Dialog Box

To perform a variety of matrix operations, from the menus choose:

Utilities
Matrix
Matrix Operations...



Matrix and Expression. To specify the target matrix, select a matrix from the list and click the Matrix button. If you want to create a new matrix, simply type it into the box.

Operation type. Select the operation type, operations, and matrix (matrices) to be used in the expression. Your expression will appear in the Expression area as you create it.

You can operate on existing matrices or create new ones. If the target matrix already exists, it is replaced by the Expression. If the matrix does not exist, then a new matrix with that name is created.

Operations. The following operations are available:

Arithmetic

Addition (+)
 Subtraction (-)
 Scalar Division (/)

Element by element multiplication (#)
 Raising each element to a power (##)

Transformation

Square root (SQR)
 Log base e (LOG)
 Log base 10 (L10)
 Exponentiation (EXP)
 Absolute value (ABS)
 Sine (SIN)
 Cosine (COS)
 Tangent (TAN)

Hyperbolic tangent (TNH)
 Log gamma (LGM)
 Integer part (INT)
 Arc sine (ASN)
 Arc cosine (ACS)
 Arc tangent (ATN)
 Arc tangent hyperbolic (ATH)
 Arc tangent (AT2)

Manipulations

Transpose (TRP)
 Sort rows (ROWSORT)
 Sort columns (COLSORT)
 Lower triangular portion copy to upper triangular portion (FOLD)
 Reshape the Matrix (SHAPE)
 Filling in missing values (FILL)
 Drop all rows containing missing data (COMPLETE)
 Dimension (DIM)
 Number of rows (NROW)

Number of columns (NCOL)
 Column vector contains the elements as lower triangular Matrix (STRING)

Matrix with the elements of vector as the lower triangular portion and missing values above the diagonal (GNIRTS (V))

Concatenate end to end (//)
 Concatenate corner to corner (/|)
 Compare two matrices element-by-element (EQUAL)

Algebraic

Multiplication (*)
 Raise to Power (**or ^)
 Inversion (INV)
 Kronecker Product (KRON)
 Determinant (DET)
 Log Determinant (LOGDET)

Trace (TRACE)
 Eigenvalues (EIGVAL)
 Pivot on Diagonal Elements (SWEEP)
 Solution of Equations (SOLVE)
 Cholesky decomposition (CHOL)

Statistical

Row Mean (ROWMEAN)
 Column Mean (COLMEAN)
 Row Standard Deviation (ROWSTD)
 Column Standard Deviation (COLSTD)

Number of missing values in columns (COLMIS)
 Number of non-missing values in rows (ROWNUM)
 Number of non-missing values in columns (COLNUM)
 Row Standardization (ROWZSC)

Row Sum (ROWSUM)
 Column Sum (COLSUM)
 Min. value in rows (ROWMIN)
 Min. value in columns (COLMIN)
 Max. value in rows (ROWMAX)
 Max. value in columns (COLMAX)
 Number of missing values in rows
 (ROWMIS)

Column Standardization (COLZSC)
 Row Rank (ROWRANK)
 Column Rank (COLRANK)
 Correlation (CORR)
 Covariance (COVA)
 Cross Product of deviations for column of
 matrix (SSCP)

Relational/Logical

Less than (<)
 Greater than (>)
 Less than or equal to (<=)
 Greater than or equal to (>=)
 Not equal to (<>)

Equal to (==)
 AND
 OR
 NEGATION (NOT)

Design

"0,1" design variables (DESIGN0)
 "1,0,-1" design variables (DESIGN1)
 Full rank design variables (DESIGNF)

Equally spaced orthogonal components
 (ORTHEQ)
 Unequally spaced orthogonal components
 (ORTHUN)

Row name(s). You can specify names for the rows of the matrix. Use commas or spaces to separate the names.

Column name(s). You can specify names for the columns of the matrix. Use commas or spaces to separate the names.

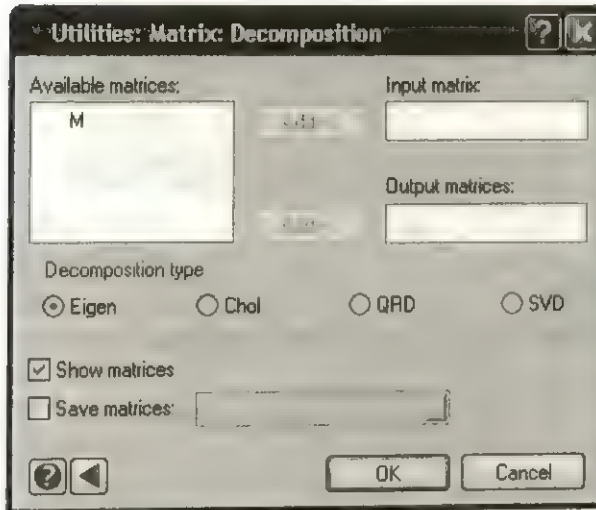
Save matrix. You can save the generated matrix to a specified file. The following options are available: Rectangular, SSCP, Covariance, Correlation, Dissimilarity and Similarity.

Show matrix. Prints the generated matrix to the current output device.

Decomposition Dialog Box

To open the Decomposition dialog box, from the menus choose:

Utilities
Matrix
Decomposition...



The following options can be specified:

Input matrix. Enter a name for the matrix you want to decompose.

Output matrices. Select names for output matrices from the list or just type names separated by space.

Decomposition type. You need to choose one from the following options:

- **Eigen.** This gives two matrices, the first consists of eigenvalues and the second, eigenvectors of the input matrix.
- **Chol.** This gives the diagonal matrix and the lower triangular matrix obtained by Cholesky decomposition of the input matrix.
- **QRD.** This decomposes the input matrix into Q and R, where the elements of R are 0 below the diagonal; that is r_{jk} is 0 whenever $j > k$.

- **SVD.** This gives the orthogonal matrix *U*, matrix of singular values *D*, and the orthogonal matrix *V* obtained in the singular value decomposition of the input matrix.

Show matrices. Print the generated matrices to the current output device.

Save matrices. You can save all output matrices to specified files.

Using Commands

For reading a matrix:

```
USE FILENAME/ MAT = name
MAT name = [a11, a12, ..., a1n; .....; am1, am2, ..., amn]
ROWNAME or COLNAME matrix = name1, name2 name3,...
SHOW matrix1, matrix2,...
SAVE name1/ MAT = name2
```

For generating matrices:

```
MAT name = GAID or DIAG or DIAG3 (argument)
MAT name = I(order)
MAT name = M(n, p, value)
ROWNAME or COLNAME matrix = name1, name2 name3,...
SHOW matrix1, matrix2,...
SAVE name1 / MAT = name2
```

For organizing matrices:

```
DIRECTORY or SHOW or CLEAR MATRIX = name1,name2,...
MSAVE name1 / MAT = name2
```


For performing row/column operations:

```
MLET result = function(varname)
MDELETE COLUMNS = columnlist
MAT name = mat_name (ref to rows; ref to columns)
MDELETE ROWS = rowlist
MSELECT condition
SHOW matrix1, matrix2,...
MSAVE name1 / MAT = name2
```

For performing matrix operations:

```
MAT result = name1 operator name2
MAT result = function (argument)
ROWNAME or COLNAME matrix = name1, name2 name3,...
SHOW matrix1, matrix2,...
MSAVE name1 / MAT = name2
```

(operator +, -, /, #, ##, *, **, ^, ||, //, /|, <, >, <=, >=, <>, =, and or, not)

(function = TRP, ROWSORT, COLSORT, FOLD, SHAPE, FILL, COMPLETE, DIM, NROW, NCOL, STRING, GNIRTS, EQUAL, INV, KRON, DET, LOGDET, TRACE, EIGVAL, SWEEP, SOLVE, CHOL, ROWMEAN, COLMEAN, ROWSTD, COLSTD, ROWSUM, COLSUM, ROWMIN, COLMIN, ROWMAX, COLMAX, ROWMIS, COLMIS, ROWNUM, COLNUM, ROWZSC, COLZSC, ROWRANK, COLRANK, CORR, COVA, SSCP, DESIGN0, DESIGN1, DESIGNF, ORTHEQ, ORTHUN, SQR, LOG, L10, EXP, ABS, SIN, COS, TAN, TNH, LGM, INT, ASN, ACS, ATN, ATH, AT2)

For decomposing matrices:

```
CALL EIGEN (or CHOL or QRD or SVD) (name1, name2, ... namen)
SHOW matrix1, matrix2,...
MSAVE name1 / MAT = name2
```


Examples

Example 1

Entering Data and Defining Matrices

Creating a Matrix

You can directly type a matrix or open a SYSTAT file as a matrix. Use the MAT command to create a matrix.

The input is:

```
MAT name = expression
```

where *name* is the name of the matrix and *expression* is either the entries of a matrix or an algebraic expression involving matrices, operators, and functions.

Typing a Matrix

To enter a matrix, type the entries separated by commas or spaces, with an open bracket before the first entry and a closed bracket after the last entry. If you are using the dialog box, then there is no need to type brackets. Separate each row with a semicolon, if you type several rows on one line. Use 'Enter' to type the next row in the dialog box. You can type each row on a separate line:

```
MAT name = [first row;  
second row;  
.  
.  
.  
last row]
```

or in one line:

```
MAT name = [first row; second row; ...; last row]
```

To enter a missing numeric value, type a period. To enter a missing character value, type a space surrounded by quotation marks.

Character Values

Most of the discussion in this chapter involves matrices with only numeric entries. A matrix can contain columns of character data. However, you cannot use most of the Matrix functions and operators on such matrices.

Naming Rows and Columns

If you save a matrix to a file, the column names are considered variable names, and the row names are saved as a variable named *ROWNAME\$*, added as the last column.

Use Column of a File to Label Rows

You can name the rows of a matrix with the entries in a column while you read a data file as matrix. This is useful for matrices with a large number of rows. Suppose that the names of the students are stored in the *NAME\$* column of the data file *GRADES*. You could name the rows of *GRADES* using *NAME\$*.

The input is:

```
USE GRADES/MAT = GRADES ROWNAME = name$
```

Requesting a Directory of Matrices

You can get a description of all the matrices known to Matrix. As an example, we request descriptions of matrices **A**, **B**, and **C**. If you do not specify a matrix name, SYSTAT displays description for all matrices known to Matrix.

The input is:

```
MAT a = [2  4 ; 3  1 ; 1  5]
MAT b = [2  3  1 ; 4  1  5]
USE LONGLEY / MAT = c
DIR a b c
```


The output is:

```

Name of the Matrix      a
Row Numbers are:       3
Row Names are:         R_1 , R_2 , R_3
Column Numbers are:    2
Column Names are:      C_1 , C_2

Name of the Matrix      b
Row Numbers are:       2
Row Names are:         R_1 , R_2
Column Numbers are:    3
Column Names are:      C_1 , C_2 , C_3

Name of the Matrix      c
Row Numbers are:      16
Row Names are:        R_1 , R_2 , R_3 ,
                     R_4 , R_5 , R_6 ,
                     R_7 , R_8 , R_9 ,
                     R_10 , R_11 , R_12 ,
                     R_13 , R_14 , R_15 ,
                     R_16
Column Numbers are:    7
Column Names are:      DEFLATOR , GNP ,
                     UNEMPLOY , ARMFORCE,
                     POPULATN , TIME ,
                     TOTAL

```

The Active Matrix

The Directory report shown above shows that the matrices **A**, **B**, and **C** are *known* to the Matrix procedure. The distinction between *active* and *known* matrices is important for the MDELETE, Rows and Columns, and MSELECT options of Row/Column Operations and also for MLET statements because they operate on the last matrix used—that is, the *active matrix*. Other commands and functions include an argument that identifies the input matrix.

Generating Matrices

Matrix can generate an identity matrix of rank n or an $n \times p$ matrix with a user-specified number. Matrix can also generate five types of design variables for categorical variables; this is discussed in “Example 7: Design Variables”.

Diagonal Matrix from a Vector

Use the GAID (DIAG spelled backwards) command to get a diagonal matrix from the specified vector. As an example, we request a diagonal matrix from the vector **V** and store the result as **V_DIAGMAT**.

The input is:

```
MAT v = [2  4  6 ]
MAT v_diagmat = GAID(v)
SHOW v_diagmat
```

The output is:

Matrix Name: v_diagmat

	C_1	C_2	C_3
R_1	2	0	0
R_2	0	4	0
R_3	0	0	6

Example 2

Element-by-Element Operations and Functions

For element-by-element operations, you can think of each matrix entry as a separate entity. SYSTAT provides a variety of such operators and functions. For the examples, we use small matrices with integer entries so that you can easily follow the calculations.

Arithmetical Operations

You can add or subtract matrices with the same dimensions (number of rows and columns) by adding or subtracting corresponding elements. Use the operator + for addition and - for subtraction. Matrix performs several variations of matrix addition and subtraction. In addition to adding and subtracting two matrices with the same dimensions element-by-element, you can:

- Add (subtract) the same number to (from) each element of a matrix.
- Add (subtract) a row vector to (from) each row of a matrix.
- Add (subtract) a column vector to (from) each column of a matrix.

As an example, we add and subtract matrices **M1**, **M2** and **M3** from the matrix **A** and store the results as **SUM1**, **SUM2**, **SUM3**, **DFFRNCE1**, **DFFRNCE2**, and **DFFRNCE3** respectively.

The input is:

```
MAT a = [1 2 3; 4 5 6]
MAT m1 = [3]
MAT m2 = [3 2 1]
MAT m3 = [3; 2]
MAT sum1 = a + m1
MAT sum2 = a + m2
MAT sum3 = a + m3
MAT dffrnce1 = a-m1
MAT dffrnce2 = a-m2
MAT dffrnce3 = a-m3
SHOW sum1 sum2 sum3 dffrnce1 dffrnce2 dffrnce3
```

The output is:

Matrix Name: sum1

	C_1	C_2	C_3
R_1	4	5	6
R_2	7	8	9

The remaining output is not listed.

Element-by-Element Multiplication and Division

Use # and / operators for element-by-element multiplication and division.

Note: Do not confuse element-by-element multiplication with matrix multiplication. Matrix multiplication is discussed in Example 4: Matrix Algebra.

In addition to scalar multiplication and division, the # and / have other uses. You can:

- Multiply or divide two matrices of the same size element-by-element.
- Multiply or divide each row of a matrix element-by-element by a row matrix.
- Multiply or divide each column of a matrix element-by-element by a column matrix.

As an example, we request multiplication and division of the matrix A by S1, S2 and S3 and store the results as **PROD1**, **PROD2**, **PROD3**, **QUOT1**, **QUOT2**, and **QUOT3** respectively.

The input is:

```

MAT a = [2  4  6; 8 10 12]
MAT s1 = [1 2 3; 4 5 6]
MAT s2 = [4; 2]
MAT s3 = [1 2 3]
MAT prod1 = a # s1
MAT prod2 = a # s2
MAT prod3 = a # s3
MAT quot1 = a/s1
MAT quot2 = a/s2
MAT quot3 = a/s3
SHOW prod1 prod2 prod3 quot1 quot2 quot3

```

The output is:

Matrix Name: prod1

	C_1	C_2	C_3
R_1	2	8	18
R_2	32	50	72

The remaining output is not listed.

Raising Each Element to a Power

The power operator (##) can be used in several different ways. You can:

- Raise each element of a matrix to the same power.
- Raise each element of a matrix to the power of the corresponding element of a second matrix with the same dimensions.
- Raise the elements of each row of a matrix to the powers of the corresponding elements of a row matrix.
- Raise the elements of each column of a matrix to the powers of the corresponding elements of a column matrix.

As an example, we use ## operator on the matrix **A** to raise powers **P1**, **P2**, **P3**, and **P4** and store the results as **HASHHASH1**, **HASHHASH2**, **HASHHASH3**, and **HASHHASH4**.

The input is:

```
MAT a = [2 4; 3 1; 1 5]
MAT p1 = [3 2; 3 7; 5 0]
MAT p2 = [4; 3; 2]
MAT p3 = [3 2]
MAT p4 = 3
MAT hashhash1 = a ## p1
MAT hashhash2 = a ## p2
MAT hashhash3 = a ## p3
MAT hashhash4 = a ## p4
SHOW hashhash1 hashhash2 hashhash3 hashhash4
```

The output is:

Matrix Name: hashhash1

	C_1	C_2
R_1	8	16
R_2	27	1
R_3	1	1

The remaining output is not listed.

Transforming Matrices

You can use SYSTAT to transform a matrix element by element. Matrix provides many of the functions available for single variables. As an example, we request a square root transform on the matrix **K** and store the result as **SQRT**.

The input is:

```
MAT k = [49 9 16 ; 4 0 36 ; 24 4 1]
MAT sqrt = SQR(k)
SHOW sqrt
```

The output is:

Matrix Name: sqrt

	C_1	C_2	C_3
R_1	7	3	4
R_2	2	0	6
R_3	5	2	1

Relational/Logical Operators

Matrix in SYSTAT provides relational operators to compare matrices with the same dimension element by element. The result is a matrix containing only 0's and 1's--0's in positions where the relation is false, and 1's where the relation is true. As an example, we compare the matrices **A** and **B** and store the result as **C**.

The input is:

```
MAT a = [2  4  6  2; 1  1  3  7; 5  3  0  6; 3  6  1  3]
MAT b = [3  5  3  2; 3  0  3  5; 9  1  3  6; 4  1  0  4]
MAT c = a < b
SHOW c
```

The output is:

Matrix Name: c

	C_1	C_2	C_3	C_4
R_1	1	1	0	0
R_2	1	0	0	0
R_3	1	0	1	0
R_4	1	0	0	1

The Matrix procedure has logical AND, OR, and NOT operators. The matrices that result from these operators are matrices of 0's (false values) and 1's (true values).

- **Logical AND.** Use AND operator to compare two matrices with the same dimensions element-by-element. The resulting matrix has 1's in positions where both the original matrices have nonzero values and 0's elsewhere.
- **Logical OR.** Use OR operator to compare two matrices with the same dimensions element-by-element. The resulting matrix has 1's in positions where at least one of the original matrices has a nonzero value, and 0's elsewhere.
- **Logical negation.** Use NOT to negate the elements of a matrix. It operates on a single matrix. The resulting matrix gets a 1 wherever the original matrix is 0, and a 0 wherever the original matrix is nonzero.
- **Exclusive OR.** You can use a combination of operators to form an exclusive OR. The resulting matrix gets a true value (1) only in positions where one matrix or the other has a true value; if both matrices have a true value, the result is false. To get **A EXCLUSIVE OR B**, request (a AND (NOT b)) OR ((NOT a) AND b).

As an example, we compare the matrices **A**, **B** and **C** with various combinations of logical operators and store the results as **R1**, **R2**, **R3**, **R4**, **R5**, **R6**, and **R7**.

The input is:

```

MAT a = [1 0 0 1; 2 1 0 1; 0 1 0 0]
MAT b = [0 1 0 1; 1 1 0 0; 0 1 0 3]
MAT c = [1 1 0 1; 0 0 1 1; 1 1 1 0]
MAT r1 = a AND b
MAT r2 = NOT a
MAT r3 = a OR b
MAT r4 = NOT a OR NOT b
MAT r5 = (a AND b) OR c
MAT r6 = b AND NOT (a OR c)
MAT r7 = (a AND (NOT b)) OR ((NOT a) AND b)
SHOW r1 r2 r3 r4 r5 r6 r7

```

The output is:

Matrix Name: r1

	C_1	C_2	C_3	C_4
R_1	0	0	0	1
R_2	1	1	0	0
R_3	0	1	0	0

The remaining output is not listed.

Example 3

Manipulating Matrices

Reordering Rows and Columns

Use ROWSORT and COLSORT functions of Manipulations to reorder the rows and columns of your matrix. As an example, we request to reorder rows and columns of the matrix **M**.

The input is:

```

MAT m = [2 4 6 2; 1 1 3 7; 5 3 0 6; 3 6 1 3]
MAT neword = ROWSORT(m, 3)
MAT neword2 = COLSORT(m, 4)
SHOW neword neword2

```


The output is:

Matrix Name: neword

	C_1	C_2	C_3	C_4
R_1	5	3	0	6
R_2	3	6	1	3
R_3	1	1	3	7
R_4	2	4	6	2

Matrix Name: neword2

	C_1	C_2	C_3	C_4
R_1	6	2	2	4
R_2	3	1	7	1
R_3	0	5	6	3
R_4	1	3	3	6

Fold

The FOLD function of Manipulations copies the lower triangular portion of a matrix to the portion of the matrix above the diagonal so that the matrix is symmetric. As an example, we fold the matrix **SQRMAT** and store the result as **FOLDED**.

The input is:

```
MAT sqrmat = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14,
15; 16 17 18 19 20; 21 22 23 24 25]
MAT folded = FOLD(sqrmat)
SHOW sqrmat folded
```

The output is:

Matrix Name: sqrmat

	C_1	C_2	C_3	C_4	C_5
R_1	1	2	3	4	5
R_2	6	7	8	9	10
R_3	11	12	13	14	15
R_4	16	17	18	19	20
R_5	21	22	23	24	25

Matrix Name: folded

	C_1	C_2	C_3	C_4	C_5
R_1	1	6	11	16	21
R_2	6	7	12	17	22
R_3	11	12	13	18	23
R_4	16	17	18	19	24
R_5	21	22	23	24	25

Reconfiguring a Matrix

You can change the dimensions of a matrix with the **SHAPE** function of Manipulations. **SHAPE** fills in the new matrix with the values of the original matrix, starting with the element in the first row and first column and working from left to right across the rows. **SHAPE** uses only as many elements as it needs. For example, if the original matrix is 4 x 5 and you specify dimensions 3 x 3 for the new matrix, **SHAPE** uses only the first nine elements for the original. Conversely, if the original matrix does not contain enough elements to fill the new matrix, **SHAPE** fills the extra positions with missing values.

Filling in Missing Values

You can use the **FILL** function of Manipulations to replace the missing values in your matrix with specified values. As an example, we fill the missing values of the matrix **A** with zeros and store the result as **NEWA**.

The input is:

```
MAT a = [1 2 3 .; 4 5 6 7; 8 9 . .; 10 . . .]
MAT newa = FILL(a, 0)
SHOW a newa
```

The output is:

Matrix Name: a

	C_1	C_2	C_3	C_4
R_1	1	2	3	0
R_2	4	5	6	7
R_3	8	9	0	0
R_4	10	0	0	0

Matrix Name: newa

	C_1	C_2	C_3	C_4
R_1	1	2	3	0
R_2	4	5	6	7
R_3	8	9	0	0
R_4	10	0	0	0

Determining the Dimensions of a Matrix

You can use **DIM**, **NROW**, and **NCOL** functions of Manipulations for determining the number of rows and columns of a matrix. The **NROW** and **NCOL** functions return the

number of rows and columns of a specified matrix. The DIM function returns both the number of rows and the number of columns as a 2 x 1 matrix.

Changing a Matrix to a Vector and Vice Versa

The STRING function of Manipulations writes the lower-triangular portion of a matrix as a vector. The GNIRTS (STRING spelled backwards) function writes a row or column vector to the lower-triangular portion of a matrix.

Concatenating Matrices

You can concatenate matrices side-by-side, end-to-end, or corner-to-corner. As an example, we request each type of concatenation from matrices **A**, **B**, **C**, and **D**.

The input is:

```
MAT a = [1; 0; 4]
MAT b = [1; 3; 5; 7]
MAT c = [2 6 8 9]
MAT d = [4 3 3; 2 3 1]
MAT concatenate = ((a || b) // c) || d
SHOW concatenate
```

The output is:

	C_1	C_1	C_1	C_2	C_3	C_6	C_7
R_1	1	1
R_2	0	3
R_3	4	5
R_4	.	7
R_1	2	6	8	9	.	3	3
R_1	2	3	1
R_2

Determining Whether Two Matrices Are Equal

The EQUAL function of Manipulations compares two matrices element-by-element and determines whether they are equal. The EQUAL function results in a scalar---a 1 if the matrices are equal or a 0 if they are not equal.

Note: Do not confuse the EQUAL function with the == operator. The EQUAL function results in a scalar Z (0 or 1). The == operator results in a matrix of 1's (where corresponding elements are equal) and 0's (where corresponding elements are

unequal). As an example, we request a test of whether the matrices **A** and **B** are equal or not.

The input is:

```
MAT a = [2 3 4; 3 4 2; 1 2 4]
MAT b = [2 3 4; 3 4 2; 1 2 5]
MAT eq = EQUAL(a, b)
SHOW eq
```

The output is:

Matrix Name: eq

	C_1
R_1	0

Example 4

Matrix Algebra

You can use SYSTAT to perform a variety of algebraic operations on matrices. Some of them are discussed here.

Raising a Matrix to a Power

Use ****** (or **^**) operator to raise a matrix to a power. As an example, we request the cube of a matrix **A** and store the result as a matrix named **A_CUBED**.

The input is:

```
MAT a = [2 1 3; 0 2 3; 5 3 2]
MAT a_cubed = a ** 3
```

Note that only square matrices can be raised to a power.

Kronecker Product

The result of the Kronecker product of $n \times p$ matrix **A** and $r \times s$ matrix **B**, contains a $r \times s$ sub-matrix for each element of **A** (that is, it has nr rows and ps columns). Each $r \times s$ sub-matrix is the product of the corresponding element in **A** with the matrix **B**. As

an example, we request Kronecker product of matrices **A** and **B** to produce the matrix **C**.

The input is:

```
MAT a = [1 2; 3 4; 5 6]
MAT b = [7 8; 9 10]
MAT c = KRON(a,b)
SHOW c
```

The output is:

Matrix Name: c

	C_1	C_2	C_3	C_4
R_1	7	8	14	16
R_2	9	10	18	20
R_3	21	24	28	32
R_4	27	30	36	40
R_5	35	40	42	48
R_6	45	50	54	60

Eigenvalues

When you multiply a matrix by a column matrix (or vector), the result is another column matrix of the same dimension. Thus, a matrix is a rotation of vectors. A non-zero vector **v** is called an **eigenvector** of a matrix **A** if there exists a scalar **s** such that:

$$Av = sv$$

A scalar **s** is called an **eigenvalue** of **A** if there exists a non-zero vector **v** such that:

$$Av = sv$$

Use the EIGVAL function to compute the eigenvalues of a matrix.

Cholesky Decomposition

The Cholesky decomposition is often used to generate data with a specific correlation structure or to standardize correlated data (for example, to compute Mahalanobis distances). The Matrix procedure provides two forms of Cholesky decomposition. The first form is discussed here, the other version of Cholesky decomposition is discussed in the Example 6 "Matrix Decomposition" on page 276.

Use the CHOL function of Matrix Algebra to get the Cholesky decomposition of a symmetric matrix. If you start with a matrix **R**, the Cholesky decomposition finds **L** such that

$$\mathbf{R} = \mathbf{L} \cdot \mathbf{L}'$$

where **L** has zeros above the diagonal. Since **L** is triangular, it is much easier to invert than **R**. Note that:

$$\begin{aligned}\mathbf{R}^{-1} &= (\mathbf{L} \cdot \mathbf{L}')^{-1} \\ &= (\mathbf{L}')^{-1} \cdot (\mathbf{L})^{-1} \\ &= (\mathbf{L}^{-1})' \cdot \mathbf{L}^{-1}\end{aligned}$$

As an example, we create the matrix **SS** using the SYSTAT data file RANSAMPLE and request its Cholesky decomposition.

The input is:

```
USE RANSAMPLE / MAT = h
MAT ss = SSCP (h)
MAT my_L_mat = CHOL(ss)
SHOW ss my_L_mat
```

The output is:

Matrix Name: ss

	C_1	C_2	C_3
R_1	92	106	211
R_2	106	202	322
R_3	211	322	666

Matrix Name: my_L_mat

	C_1	C_2	C_3
R_1	10	0	0
R_2	11	9	0
R_3	22	9	10

Example 5

Statistical Functions

We use the GRADES data file to demonstrate some of the statistical functions. First we use some column functions.

The input is:

```
USE GRADES/MAT = grades
ROWNAME grades = Mark Cindy Jeff Greg Michele Nicky
MAT count = COLNUM (grades)
MAT test_avg = COLMEAN (grades)
MAT testhigh = COLMAX (grades)
MAT test_low = COLMIN (grades)
SHOW count test_avg testhigh test_low
```

The output is:

Matrix Name: count

	C_1	C_2	C_3	C_4	C_5	C_6
R_1	6	5	6	5	6	6

Matrix Name: test_avg

	C_1	C_2	C_3	C_4	C_5	C_6
R_1	79	84	83	90	84	91

Matrix Name: testhigh

	C_1	C_2	C_3	C_4	C_5	C_6
MAXIMUM	97	90	95	100	95	97

Matrix Name: test_low

	C_1	C_2	C_3	C_4	C_5	C_6
R_1	59	77	73	81	64	75

The resulting matrices have an entry for each column of the original matrix. The **COUNT** matrix indicates that there is a missing score for both the second and third quiz. The **TEST_AVG** matrix shows that the lowest average score occurred on the first quiz, and the highest average score was for the final. **TESTHIGH** and **TEST_LOW** show the highest and lowest scores on each test.

Now let us try some row functions and concatenate the resulting columns (matrices):

```
MAT count = ROWNUM(grades)
MAT stu_avg = ROWMEAN(grades)
MAT stu_high = ROWMAX(grades)
MAT stu_low = ROWMIN(grades)
MAT results = count || stu_avg || stu_high || stu_low
ROWNAME results = Mark Cindy Jeff Greg Michele Nicky
COLNAME results=count stu_avg stu_high stu_low
SHOW results
```


The output is:

Matrix Name: RESULTS

	count	stu_avg	stu_high	stu_low
Mark	6.000	88.833	95.000	77.000
Cindy	6.000	94.000	100.000	85.000
Jeff	5.000	70.400	81.000	59.000
Greg	6.000	85.167	92.000	78.000
Michele	5.000	81.800	95.000	67.000
Nicky	6.000	87.333	93.000	82.000

The output provides information about rows (students). The first result (non-missing) is the number of tests taken by each student. The second is the average of the non-missing test scores for each student. The third and fourth panels give each student's high and low grade, respectively. Each student's name was specified using the function ROWNAME. Matrix knows the contents of each column, and it assigns the respective name.

Correlation, Covariance, and SSCP Matrices

SYSTAT has functions for computing covariance, correlation and cross product of deviations (SSCP) matrices in Statistical functions of Matrix Operations. Given an $n \times p$ matrix,

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

the cross product of deviations, covariance, and correlation matrices of X are defined, respectively, as:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pp} \end{bmatrix} \quad S = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1p} \\ s_{21} & s_{22} & \cdots & s_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{p1} & s_{p2} & \cdots & s_{pp} \end{bmatrix} \quad R = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1p} \\ r_{21} & r_{22} & \cdots & r_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ r_{p1} & r_{p2} & \cdots & r_{pp} \end{bmatrix}$$

for all $i, j = 1, \dots, p$:

$$a_{ij} = \sum_{k=1}^n (x_{ki} - \bar{x}_i)(x_{kj} - \bar{x}_j)$$

$$s_{ij} = a_{ij} / (n-1)$$

$$r_{ij} = \frac{s_{ij}}{\sqrt{s_{ii} \cdot s_{jj}}}$$

Correlations for Sets of Variables

As an example, let us look at correlations between all pairs of variables in the Longley data (*TIME* is omitted).

The input is:

```
USE LONGLEY/ MAT = longley
MAT xvars = longley (;DEFLATOR .. POPULATN)
MAT corrx = CORR(xvars)
SHOW corrx
```

The output is:

Matrix Name: corrx

	C_1	C_2	C_3	C_4	C_5
R_1	1.000	0.992	0.621	0.465	0.979
R_2	0.992	1.000	0.604	0.446	0.991
R_3	0.621	0.604	1.000	-0.177	0.687
R_4	0.465	0.446	-0.177	1.000	0.364
R_5	0.979	0.991	0.687	0.364	1.000

Correlations for Two Sets of Variables

When there are many variables, the correlations matrix can be very large and hard to view—especially if you are interested only in correlations among one group of

variables against another. Here, define *DEFLATOR* and *GNP* as one set and *UNEMPLOY*, *ARMFORCE*, and *POPULATN* as a second set.

```
USE LONGLEY / MAT = longley
MAT set_a=longley( ; DEFLATOR, GNP)
MAT set_b=longley( ; UNEMPLOY .. POPULATN)
MAT sda=SQR (DIAG(SSCP (set_a)))
MAT sdb=SQR(DIAG(SSCP(set_b)))
MAT corr_ab=TRP((set_a-COLMEAN(set_a))/sda)*((set_b-,
COLMEAN(set_b))/sdb)
SHOW corr_ab
```

The output is:

Matrix Name: corr_ab

	UNEMPLOY	ARMFORCE	POPULATN
deflator	0.621	0.465	0.979
gnp	0.604	0.446	0.991

Example 6

Matrix Decomposition

SYSTAT provides Cholesky, QR, and Singular value decomposition options. It also computes eigenvectors.

Eigenvectors

Use the *EIGEN* function of the Matrix decomposition to get eigenvalues as well as eigenvectors. As an example, we compute the eigenvectors and eigenvalues of the matrix *A* and store the results as *EIGVALS* and *EIGVECT* respectively.

The input is:

```
MAT b = [2 1 5 3; 1 1 3 6; 5 3 0 1; 3 6 1 3]
MAT a = b+TRP(b)
CALL EIGEN (eigvals, eigvect, a)
FORMAT 3
SHOW eigvals eigvect
```


The output is:

Matrix Name: eigvals

	C_1	C_2	C_3	C_4
R_1	22.397	0.000	0.000	0.000
R_2	0.000	5.979	0.000	0.000
R_3	0.000	0.000	-4.155	0.000
R_4	0.000	0.000	0.000	-12.221

Matrix Name: eigvect

	C_1	C_2	C_3	C_4
R_1	0.468	0.620	0.451	0.440
R_2	0.515	-0.390	-0.532	0.548
R_3	0.400	0.486	-0.543	-0.556
R_4	0.597	-0.476	0.469	-0.445

Cholesky

The alternative version of the Cholesky decomposition is:

$$\mathbf{R} = \mathbf{L}^* \cdot \mathbf{D} \cdot \mathbf{L}^{*'}$$

where \mathbf{L}^* has zeros above the diagonal and \mathbf{D} is 0 everywhere but the diagonal. Some prefer this decomposition to the one shown in the Example 4 "Matrix Algebra" on page 270 because square roots are not computed. Note that in the previous form

$$\mathbf{R} = \mathbf{L} \cdot \mathbf{L}'$$

implies that

$$\mathbf{L} = \mathbf{L}^* \cdot \sqrt{\mathbf{D}}$$

QR

For an $n \times p$ matrix \mathbf{X} , the QR decomposition is

$$\mathbf{X} = \mathbf{Q} \cdot \mathbf{R}$$

where \mathbf{Q} is an $n \times n$ matrix and \mathbf{R} is an $n \times p$ matrix such that:

$$\mathbf{Q}' \cdot \mathbf{Q} = \mathbf{I} \text{ and } \mathbf{Q} \cdot \mathbf{Q}' = \mathbf{I}$$

The elements of \mathbf{R} are 0 below the diagonal, i.e., r_{jk} is 0 if $j > k$.

This decomposition can make regression computations easier. Note that

$$\mathbf{Y} = \mathbf{X} \cdot \boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

$$Q' \cdot Y = R \cdot \beta + \varepsilon$$

Therefore, by starting with this structure, the computation is easy via a “back solution”.

Singular Value

Both the QR and SVD (singular value decomposition) methods can be used to find eigenvalues and eigenvectors. For an $n \times p$ matrix X , the SVD decomposition is

$$X = U \cdot D \cdot V$$

where U is an $n \times n$ orthogonal matrix, D is an $n \times p$ matrix that is 0 except for its diagonal, and V is a $p \times p$ orthogonal matrix. Here the diagonal elements of D are $d_{11}, d_{22}, \dots, d_{pp}$ when p is less than or equal to n , and $d_{11}, d_{22}, \dots, d_{nn}$ when n is less than p . In addition:

$$U \cdot U' = I_n$$

$$V \cdot V' = I_p$$

As an example, we request Cholesky and QR decompositions of the matrix SS and store the results as **MY_D_MAT**, **MY_LSTAR**, **MY_Q_MAT**, and **MY_R_MAT** respectively.

The input is:

```
USE RANSAMPLE / MAT = h
MAT ss = SSCP(h)
CALL CHOL (my_D_mat, my_Lstar, ss)
CALL QRD (my_Q_mat, my_R_mat, ss)
SHOW my_D_mat my_Lstar my_Q_mat my_R_mat
```

Use SV option of the Matrix decomposition to get singular value decomposition. As an example, we request the Singular value decomposition of the matrix A and store the results as **MY_U_MAT**, **MY_D_MAT**, **MY_V_MAT** and **A** respectively.

The input is:

```
MAT a = [1 2; 3 4; 5 6]
CALL SVD (my_U_mat, my_D_mat, my_V_mat, a)
SHOW my_U_mat my_D_mat my_V_mat a
```


Example 7

Design Variables

If you are using Matrix to request an analysis of variance or if you need to generate a set of design variables for each categorical variable in a regression model, use DESIGN0, DESIGN1, DESIGNF, ORTHEQ, or ORTHUN functions of Design type. To generate design variables, the cases in your data file need no special ordering. For example, if you have a data file named MYDATA with the variables CITY\$ (Los Angeles, Chicago, and New York) and DOSE\$ (zero, low, medium, and high) and want to generate the "1, 0, -1" type variables for CITY\$ and the orthogonal coefficients for DOSE\$, use DESIGN1 and ORTHEQ functions to get the desired results.

The input is:

```
MAT mydata = ['Los Angeles'; Chicago; 'New York'] || [zero;
low; medium; high]
COLNAME mydata = city$ dose$
MAT my_design = DESIGN1 (mydata {1 .. 3; city$}) || ,
ORTHEQ (mydata (; dose$))
SHOW my_design
```

The DESIGN1 function generates two design variables (A and B) for values of CITY\$; ORTHEQ generates three design variables (C, D, and E) for levels of DOSE\$. The following are the values generated for specific cities and doses:

CITY	A	B	DOSE	C	D	E
LA	1	0	zero	-3	1	-1
Chicago	0	1	low	-1	-1	3
New York	-1	-1	medium	1	-1	-3
			high	3	1	1

Actually, the values of the orthogonal components are not quite what was stated here. SYSTAT normalizes each component by dividing by the square root of the sum of the squares of the coefficients (for example, for the design variable C, the value for "zero dose" is

$$\frac{3}{\sqrt{9+1+1+9}}$$

or -0.671). The following are the actual values:

DOSE	C	D	E
Zero	-0.671	0.500	-0.224

Low	-0.224	-0.500	0.671
Medium	0.224	-0.500	-0.671
High	0.671	0.500	0.224

Use of ORTHUN in Regression Analysis

The generated observations are stored in the file named REGORTH0. The aim is to explore the polynomial regression

$$Y = a + bx + cx^2 + dx^3$$

to determine whether the terms with higher degrees are necessary or redundant.

The input is:

```
USE REGORTH0 / MAT = d
Mat x = d (;1)
MAT y = d ( ;2)
MAT xt = x (1; 1)
MAT xre = x / xt
MAT q = ORTHUN (xre, xre)
MAT z = TRP (y) * q
MAT s = z # z
MAT ssub = s (1; 1, 2, 3)
SHOW ssub
```

The output is:

Matrix Name: ssub

	C_1	C_2	C_3
Y	1671.255	33.294	4.099

The three terms, from right to left, in ssub are the sum of squares corresponding to the cubic, quadratic and linear regression coefficients.

```
MAT ss = SSCP(y)
SHOW ss
```

The output is:

Matrix Name: ss

	C_1
R_1	1727.983

Here **SS** gives the total sum of squares (adjusted for the constant). Now the significance can be successively tested using F-tests to decide whether to retain the higher order terms, and thus a decision regarding the appropriate degree for the polynomial regression can be taken. (Indeed one finds that the 24 elements of the matrix **S** is a breakup of the total sum of squares into 24 components, corresponding to the coefficients in the polynomial equation of degree 24.) One has to take care that while using the **ORTHUN** function all the elements of the vector **x** have to be distinct with the first element equal to one. Also if there are too many elements in vector **x**, **SYSTAT** may find the problem difficult to handle. That is why a vector **xre** was created from the vector **x**.

Example 8 **Packing Two Records into One**

Sometimes, you may have to restructure your data for a statistical analysis. For example, suppose you record the age and blood cholesterol levels for two groups of women. Women in the first group use contraceptive pills; women in the second group do not. The data in the **MYSTUDY** file look like this:

PILL	AGE	CHOL
1	25	200
2	25	211
1	33	230
2	33	243
1	19	180
2	19	215
1	39	215
2	39	175
1	28	189
2	28	163
1	20	179
2	20	175
1	35	300
2	35	224

A **PILL** value of 1 indicates that the woman takes the pill; a value of 2 indicates that she does not. You want to use a matched pairs *t*-test to test the hypothesis that the mean difference in blood cholesterol between women who take the pill and women who do not is 0. First, using the **SHAPE** function, match a woman who takes the pill with a woman of the same age who does not:

```
MAT matched = SHAPE(cholesterol,7,6)
```


or

```
MAT matched = SHAPE(cholesterol, NROW(cholesterol)/2,
  NCOL(cholesterol)*2)
```

Now the data file looks like this:

1	25	200	2	25	211
1	33	230	2	33	243
1	19	180	2	19	215
1	39	215	2	39	175
1	28	189	2	28	163
1	20	179	2	20	175
1	35	300	2	35	224

Each case has the cholesterol value for a pill user and for her age-matched control. You can drop all of the columns except the two that contain cholesterol and rename them *P_CHOL* and *NOP_CHOL*, respectively:

```
MDELETE COLUMNS = 1, 2, 4, 5
COLNAME matched = P_CHOL, NOP_CHOL
MSAVE matched
```

Now you can request the matched pairs *t*-test:

```
TESTING
  USE MATCHED
  TTEST P_CHOL, NOP_CHOL
```

Example 9

Writing a Correlation Matrix as a Vector

As an example of the *STRING* function, we examine correlations among 16 variables from the *USSTATES* file. These variables include death rates from nine causes, crime rates, median household income and other economic indicators, weather information, and health care data.

You can compute the correlation matrix in *Matrix* or *Correlations*. We computed the correlation matrix, named it *USCORR*, and then used the *STRING* function to write the correlations as a vector.

The following are the variable names:

ACCIDENT	CARDIO	CANCER	PULMONAR	PNEU_FLU	DIABETES
LIVER	VIOLRATE	PROPRATE	AVGPAY	TEACHERS	TCHRSAL
MARRIAGE	DIVORCE	HOSPITAL	DOCTOR		

You can compute the correlation matrix in Matrix or Correlations. We computed the correlation matrix, named it *USCORR*, and then used the *STRING* function to write the correlations as a vector.

The input is:

```
USE USCORR / MAT = uscorr
MAT corrstr = STRING (uscorr,0)
MSAVE CORRSTR
FORMAT 6 2
SHOW uscorr corrstr
FORMAT
```

The output is:

AVGPAY	TEACHERS	ACCIDENT	CARDIO	CANCER	PULMONAR	PNEU_FLU	DIABETES	LIVER	VIOLRATE	PROP
	R_1	1.00
	R_2	-0.15	1.00
	R_3	-0.17	0.89	1.00
	R_4	0.24	0.12	0.29	1.00
	R_5	-0.16	0.47	0.33	0.24	1.00
	R_6	0.03	0.53	0.63	0.13	-0.06	1.00	.	.	.
	R_7	-0.16	0.05	0.27	0.21	-0.31	0.15	1.00	.	.
	R_8	0.00	0.06	0.10	-0.24	-0.23	-0.14	0.53	1.00	.
	R_9	-0.01	-0.37	-0.25	-0.11	-0.43	-0.34	0.41	0.68	1.00
	R_10	-0.60	0.09	0.16	-0.30	-0.09	-0.05	0.50	0.57	0.
1.00	R_11	0.24	-0.08	-0.22	0.07	0.26	0.03	-0.48	-0.58	-0.
-0.47	R_12	-0.62	0.08	0.24	-0.22	-0.11	0.07	0.53	0.49	0.
0.90	R_13	-0.52	-0.14	-0.03	0.38	-0.19	-0.24	0.40	0.09	0.
-0.01	R_14	-0.26	-0.27	-0.17	0.53	-0.25	-0.24	0.19	0.06	0.
-0.29	R_15	-0.12	0.23	-0.05	0.20	0.37	-0.12	-0.43	-0.60	-0.
-0.70	R_16	0.69	-0.65	0.14	0.25	0.06	0.06	0.44	0.44	0.
0.77		-0.37								
	TCHRSAL	MARRIAGE	DIVORCE	HOSPITAL	DOCTOR					
	R_1
	R_2
	R_3
	R_4
	R_5
	R_6
	R_7
	R_8
	R_9
	R_10
	R_11
	R_12	1.00
	R_13	0.00	1.00
	R_14	-0.36	0.73	1.00
	R_15	-0.65	-0.07	0.02	1.00
	R_16	0.81	-0.14	-0.48	-0.55	1.00

Matrix Name: corrstr

	C_1
R_1	-0.15
R_2	-0.17
R_3	0.89
R_4	0.24
R_5	0.12
R_6	0.29
R_7	-0.16
R_8	0.47
R_9	0.33
R_10	0.24
R_11	0.03
R_12	0.53
R_13	0.63
R_14	0.13
R_15	-0.06
*	
*	
R_106	-0.65
R_107	0.14
R_108	0.25
R_109	-0.23
R_110	0.06
R_111	0.06
R_112	0.44
R_113	0.44
R_114	0.21
R_115	0.77
R_116	-0.37
R_117	0.81
R_118	-0.14
R_119	-0.48
R_120	-0.55

The following is a stem-and-leaf diagram of the 210 correlations:

```
USE CORRSTR
CLSTEM C_1
```

The output is:

Stem and Leaf Plot of Variable: C_1, N = 120

```
Minimum      : -0.700
Lower Hinge  : -0.234
Median       : 0.025
Upper Hinge  : 0.258
Maximum      : 0.900
```

```

-6  944100
-5  842
-4  8886332
-3  77541
-2 H 9975554332221
-1  97766644331110
-0  9776543110
```



```

0 M 0133456666677899
1   13334699
2 H 0133334566889
3   23789
4   1347799
5   22336
6   388
7   37
8   08
9   0

```

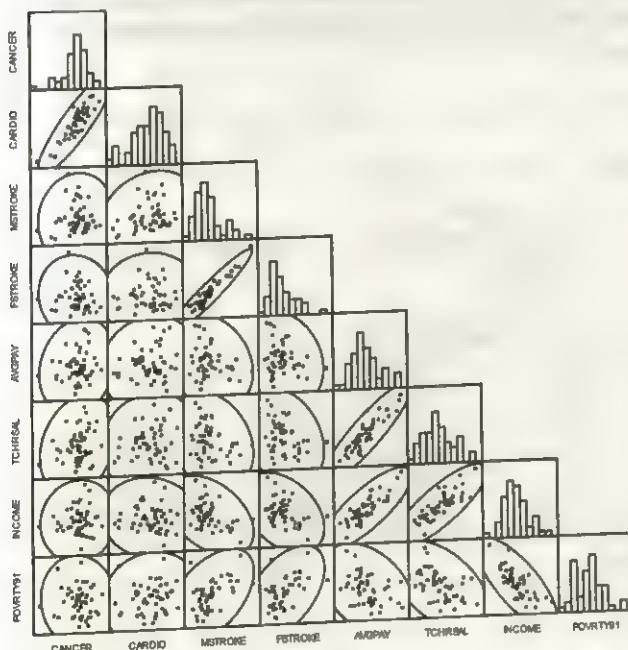
Let us use a SPLOM (scatterplot matrix) with 90% confidence ellipses to examine the bivariate distributions of the variables with sizable correlations (-0.74 , 0.80 , 0.82 , 0.91 , 0.91 , and 0.94).

```

USE USSTATES
SPLOM CANCER CARDIO MSTROKE FSTROKE,AVGPAY TCHRSAL INCOME,
POVRTY91 / HALF ELL = .99

```

The output is:



Five of the positive correlations are much larger than the others. Two of these involve death rates due to *CANCER*, *CARDIO*, *MSTROKE* (for males), and *FSTROKE* (for females). The other three involve economic indicators: *AVGPAY* (average pay), *TCHRSAL* (average salary of teachers), and *INCOME* (median family income). One

negative correlation, *POVRTY91* (percentage of the population living below the poverty level) and *INCOME*, is fairly large.

Example 10

Ridge Regression

Ridge regression is a technique that tames the estimates of the regression coefficients when severe multicollinearity exists among the independent variables. The formula for the estimated coefficients is

$$\hat{\beta} = (Z'Z + \lambda I)^{-1} Z'y$$

where *Z* is the matrix of standardized independent variables, *y* is the vector of the standardized dependent variable, and *I* is the identity matrix. The usual least-squares estimate is obtained when λ is 0. Before selecting a value for λ , researchers usually try several values and plot the resulting estimates to get an indication of their stability. Then, they choose a value of λ for which the coefficients smooth out and no longer make sudden changes.

The input is:

```
USE CEMENT / MAT = cement
FORMAT 11 7
MAT y = cement(;HEAT)
MAT x = cement(;INGREDIENT1 .. INGREDIENT4)
MAT y = COLZSC(y)
MAT x = COLZSC(x)
MAT cpdx = SSCP(x)
MAT beta = INV(cpdx)*TRP(x)*y
SHOW beta
```

The output is:

Matrix Name: beta

	C_1
R_1	0.6065120
R_2	0.5277056
R_3	0.0433897
R_4	-0.1602874

The input is:

```

MAT ridge = GAID(DIAG(cpdX))
MAT betar = INV(cpdX + ridge # 0.01)*TRP(x)*y
MAT beta = beta || betar
MAT betar = INV(cpdX + ridge # 0.02)*TRP(x)*y
MAT beta = beta || betar
MAT betar = INV(cpdX + ridge # 0.03)*TRP(x)*y
MAT beta = beta || betar
MAT betar = INV(cpdX + ridge # 0.04)*TRP(x)*y
MAT beta = beta || betar
MAT betar = INV(cpdX + ridge # 0.05)*TRP(x)*y
MAT beta = beta || betar
MAT betar = INV(cpdX + ridge # 0.06)*TRP(x)*y
MAT beta = beta || betar
MAT betar = INV(cpdX + ridge # 0.07)*TRP(x)*y
MAT beta = beta || betar
MAT betar = INV(cpdX + ridge # 0.08)*TRP(x)*y
MAT beta = beta || betar
MAT betar = INV(cpdX + ridge # 0.09)*TRP(x)*y
MAT beta = beta || betar
MAT beta = TRP(beta)
MAT RIDGE = [0.0; 0.01; 0.02; 0.03; 0.04; 0.05; 0.06; 0.07; ,
             0.08; 0.09]
MAT beta = beta || RIDGE
SHOW beta
COLNAME beta=INGREDIENT1 INGREDIENT2 INGREDIENT3 INGREDIENT4,
          RIDGE
MSAVE betal / MAT = beta
FORMAT

```

The output is:

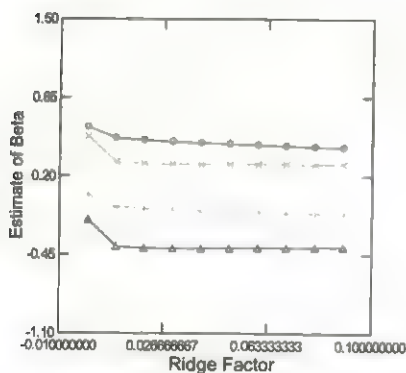
Matrix Name: beta

	R_1	R_2	R_3	R_4	C_1
R_1	0.6065120	0.5277056	0.0433897	-0.1602874	0.0000000
R_2	0.5142731	0.3166387	-0.0549317	-0.3815658	0.0100000
R_3	0.4974812	0.3032684	-0.0694199	-0.3941998	0.0200000
R_4	0.4853903	0.2996831	-0.0788427	-0.3963703	0.0300000
R_5	0.4751309	0.2986033	-0.0863759	-0.3957766	0.0400000
R_6	0.4659874	0.2984219	-0.0928003	-0.3941320	0.0500000
R_7	0.4576619	0.2985967	-0.0984335	-0.3920241	0.0600000
R_8	0.4499908	0.2989030	-0.1034462	-0.3897031	0.0700000
R_9	0.4428675	0.2992370	-0.1079470	-0.3872896	0.0800000
R_10	0.4362153	0.2995478	-0.1120128	-0.3848468	0.0900000

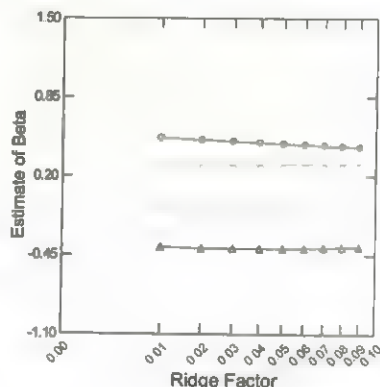
Plotting Ridge Estimates

The following is a plot of the results:

```
USE BETA1
CUT ROWNAMES$
BEGIN
PLOT INGREDIENT1..INGREDIENT4 * RIDGE / OVERLAY LINE XMIN = -.01,
XMAX = .1 YMIN = -1.1 YMAX = 1.5 XLABEL = 'Ridge Factor',
YLABEL = 'Estimate of Beta' LEGEND = NONE
PLOT INGREDIENT1..INGREDIENT4 * RIDGE/OVERLAY SMOOTH = SPLINE,
YMIN = -1.1 YMAX = 1.5 XPOW XLABEL = 'Ridge Factor',
YLABEL = 'Estimate of Beta' LOC = 6IN, 0IN,
LEGEND = -2.5IN, -2.3IN
END
```



○ INGREDIENT1
 □ INGREDIENT2
 ◇ INGREDIENT3
 △ INGREDIENT4



In the plot on the right, we added XPOW and SMOOTH = SPLINE.

Example 11 The Sweep Function

The SWEEP function is extremely useful in many statistical procedures. To operate the SWEEP function the underlying matrix has to be a square matrix and zero should not appear as the pivoting or sweeping element anywhere during the operation. If pivoting is done to some elements, the result is a partially inverted matrix. The vector in the SWEEP argument contains 0 or 1 for each element of the diagonal of the input matrix,

where '0' indicates no pivoting and '1' indicates pivoting. As an example, we use the sweep function on matrix **P** and store the result as **MY_SWEEP**.

The input is:

```
MAT p = [2 1 1; 1 2 1; 2 3 4]
MAT my_sweep = SWEEP(p, [0 1 0])
SHOW my_sweep
```

The output is:

Matrix Name: my_sweep

	C_1	C_2	C_3
R_1	1.500	-0.500	0.500
R_2	0.500	0.500	0.500
R_3	0.500	-1.500	2.500

The SWEEP function has transformed **P** into **MY_SWEEP** by using the indicated diagonal elements for sweeping. The vector in the above argument is [0 1 0], the SWEEP function pivots the second diagonal element of the input matrix **P**.

SWEEP can find the inverse. As an example, we invert the matrix **K** and store the result as **K_INV**.

The input is:

```
MAT k = [1 2 3; 2 3 4; 3 4 6]
MAT k_inv = SWEEP(k, [1 1 1])
SHOW k_inv
```

The output is:

Matrix Name: k_inv

	C_1	C_2	C_3
R_1	-2.000	0.000	1.000
R_2	0.000	3.000	-2.000
R_3	1.000	-2.000	1.000

In general if,

$$\mathbf{W} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

an $n \times n$ matrix with A as a sub-matrix of order $m \times m$ then

$$S = \text{SWEEP}(W, [1..1, 0..0])$$

with m 1's and $n-m$ 0's in the places as indicated, the SWEEP function changes W to the $n \times n$ matrix S given by

$$S = \begin{bmatrix} A^{-1} & A^{-1}B \\ -CA^{-1} & D - CA^{-1}B \end{bmatrix}$$

The SWEEP operator is the workhorse of computational statistics. Now let us see some applications of SWEEP in statistical analysis.

From a computational standpoint, the SWEEP function is useful in getting regression coefficients in the model

$$Y = X\beta + \varepsilon$$

Consider the matrix

$$W = \begin{bmatrix} X'X & X'Y \\ Y'X & Y'Y \end{bmatrix}$$

where X is an $n \times m$ design matrix.

A single application of SWEEP on matrix W permits the simultaneous computation of $\hat{\beta}$ and error sum of squares (ESS).

$$S = \begin{bmatrix} (X'X)^{-1} & \hat{\beta} \\ -\hat{\beta}' & \text{ESS} \end{bmatrix}$$

where $\hat{\beta} = (X'X)^{-1}X'Y$ and $\text{ESS} = Y'Y - Y'X(X'X)^{-1}X'Y$.

Let us demonstrate some uses of SWEEP through examples.

Evaluating a Quadratic Form

We are often required to carry out computations involving expressions of the form:

$$(\bar{x} - \bar{y})' A^{-1} (\bar{x} - \bar{y})$$

Such computations, for example, are required while computing Hotelling's T^2 statistics in multivariate analysis or sums of squares in hypothesis testing problems.

Consider

$$W = \begin{bmatrix} A & \bar{x} - \bar{y} \\ \bar{x} - \bar{y} & 0 \end{bmatrix}$$

an $n \times n$ matrix, and use the SWEEP function with choice of the vector being $[1 \ 1 \ 1 \dots 1, 0]$, 0 only in the last place. The element in the last row, last column of the resulting matrix will be

$$-(\bar{x} - \bar{y})' A^{-1} (\bar{x} - \bar{y})$$

giving us the value of the required quadratic form with an altered sign. For example,

```
MAT A = [1 1 1; 1 2 1; 1 1 2]
MAT b = [2; 3; 4]
MAT w = A || b // (trp(b) || [0])
MAT s = SWEEP(w, [1 1 1 0])
MAT q = - s(4; 4)
SHOW q
```

The output is:

Matrix Name: q

```
-----+-----
      | R_4
-----+-----
R_4 | 9.000
```

Q gives the value of $b' A^{-1} b$.

This procedure can be used to evaluate a quantity like the one above, needed to compute Hotelling's T^2 .

Solving the System

$$\mathbf{W}\mathbf{x} = \mathbf{0}$$

where \mathbf{W} is $n \times n$ singular matrix with $\text{rank}(\mathbf{W}) = n-1$.

Assuming that the $(n-1) \times (n-1)$ sub-matrix of \mathbf{W} lying in the upper left hand corner is non-singular and writing \mathbf{W} as

$$\mathbf{W} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c}' & d \end{bmatrix}$$

notice that $\begin{bmatrix} \mathbf{A}^{-1}\mathbf{b} \\ -1 \end{bmatrix}$ is a solution which can be obtained through the SWEEP function.

Now the above can be used to obtain the stationary distribution of a Markov chain.

For a Markov chain with a one-step transition probability matrix \mathbf{P} , a stationary distribution is given by a vector \mathbf{u} which satisfies

$$\mathbf{u}'\mathbf{P} = \mathbf{u}'$$

with all elements of \mathbf{u} non-negative, and adding up to one. For an irreducible aperiodic Markov chain, \mathbf{u} is unique. Using SYSTAT matrix commands, one can find this unique stationary distribution.

As an example, we request the stationary distribution of a Markov chain having a transition probability matrix \mathbf{P} and store the result as \mathbf{u} .

The input is:

```

MAT p = [0.1, 0.2, 0.2, 0.1, 0.4;
          0.1, 0.2, 0.3, 0.4, 0;
          0, 0.1, 0.2, 0.3, 0.4;
          0.4, 0.1, 0.1, 0, 0.4;
          0.2, 0.2, 0.2, 0.2, 0.2]
MAT i = i(5)
MAT w = i - p
MAT s = SWEEP(w, [1 1 1 1 0])
MAT c = s (5; 1, 2, 3, 4) || [1]
MAT Csum = ROWSUM(c)
MAT u = c/csum
SHOW u

```

The output is:

Matrix Name: u

	c_1	c_2	c_3	c_4	c_5
R_1	0.167	0.161	0.196	0.196	0.280

The matrix (vector) **U** obtained is the stationary distribution.

Solving Two Systems of Equations

Consider the problem of obtaining the vectors x and y satisfying the equations

$$ax = b \text{ and } a'y = c,$$

where a is a non-singular matrix. Let

$$S = \begin{bmatrix} a & b \\ -c' & 0 \end{bmatrix}$$

To illustrate this, consider the following equations:

$$x + y + z = 4$$

$$x - y + z = 2$$

$$x - y - z = 0$$

and, simultaneously, also the system:

$$u + v + w = 10$$

$$u - v - w = 2$$

$$u + v - w = 6$$

The input is:

```
MAT s=[1 1 1 4; 1 -1 1 2; 1 -1 -1 0; -10 -2 -6 0]
MAT sol = SWEEP(s, [ 1 1 1 0 ])
MAT sol1 = sol(1, 2, 3 ; 4)
MAT sol2 = sol(4; 1 2 3)
SHOW sol sol1
MAT sol21 = TRP(sol2)
SHOW sol21
```

The output is:

Matrix Name: sol

	C_1	C_2	C_3	C_4
R_1	0.500	0.000	0.500	2.000
R_2	0.500	-0.500	0.000	1.000
R_3	0.000	0.500	-0.500	1.000
R_4	6.000	2.000	2.000	28.000

Matrix Name: sol1

	C_4
R_1	2.000
R_2	1.000
R_3	1.000

Matrix Name: sol21

	R_4
C_1	6.000
C_2	2.000
C_3	2.000

SOL1 and **SOL21** are the solutions to the above two systems of equations.

Finding Multiple Correlation Coefficient

The example below demonstrates how the SWEEP command can be used to find the multiple correlation coefficient. You can find the multiple correlation coefficient between X and (Y Z) as follows.

The input is:

```
USE AKIMA / MAT = h
MAT r = CORR( h )
MAT w = SWEEP(r, [0 1 1])
MAT w1 = w(1; 1)
MAT mult_r = SQR(1 - w1)
SHOW mult_r
```

The output is:

Matrix Name: mult_r

	C_1
R_1	0.877

Example 12

Multiple Regression and Canonical Correlation

This example illustrates how Matrix in SYSTAT can be used to compute canonical correlation coefficients and multiple regressions. For this the IRIS data set is used. This data set consists of observations on sepal length, sepal width, petal length and petal width for three species, with 50 observations on each species.

The input is:

```
USE IRIS / MAT = iris
MAT rr = CORR( iris )
MAT r = rr(2 .. 5; 2 .. 5)
SHOW r
```

The output is:

Matrix Name: r

	C_2	C_3	C_4	C_5
R_2	1.000	-0.118	0.872	0.818
R_3	-0.118	1.000	-0.428	-0.366
R_4	0.872	-0.428	1.000	0.963
R_5	0.818	-0.366	0.963	1.000

Although apparently the Matrix *C* looks symmetric because of approximations involved in the computing process, it no longer remains exactly symmetric. Therefore the FOLD command is used.

It is to be noted that since SYSTAT computes characteristic roots of only symmetric matrices, r11 is decomposed using the CHOL function. The characteristic roots of the

C matrix and those of $(r_{11})^{-1}(r_{12})(r_{22})^{-1}(r_{21})$ are the same and are squares of canonical correlation coefficients. Characteristic roots are obtained using the EIGVAL function.

```
MAT d = EIGVAL(cc)
SHOW d
```

The output is:

Matrix Name: d

	C_1
R_1	0.885
R_2	0.015

Matrix Name: can_corr

	C_1
R_1	0.941
R_2	0.124

These are the two canonical correlation coefficients between the two sets of variables (sepal length, sepal width) and (petal length, petal width).

Now the coefficients of regression of (sepal length, sepal width) on (petal length, petal width) are to be found. To remove the first column 'species' from the Iris data, the MDELETE function is used:

```
USE IRIS / MAT = iris
MDELETE COLUMNS=1
MAT ss = SSCP(iris)
SHOW ss
```

The output is:

Matrix Name: ss

	C_1	C_2	C_3	C_4
R_1	102.168	-6.323	189.873	76.924
R_2	-6.323	28.307	-49.119	-18.124
R_3	189.873	-49.119	464.325	193.046
R_4	76.924	-18.124	193.046	86.570

The output is:

Matrix Name: b

	R_1	C_1	C_2
R_1	4.191	0.542	-0.320
R_2	3.587	-0.257	0.364

Beta (**b**) is the matrix of regression coefficients of (sepal length, sepal width) on (petal length, petal width) and beta0 (**b**₀) is the constant coefficient vector.

Now the conditional covariance matrix of one set of variables given the other set assuming multivariate normality can be computed. In the following, the conditional variances of sepal length and sepal width on petal length, petal width is found for the Iris data.

```
MAT condvar = s11_2 / 147
SHOW condvar
```

The output is:

Matrix Name: condvar

	C_1	C_2
R_1	0.162	0.099
R_2	0.099	0.152

This gives the conditional variances of (petal length, petal width) given (sepal length, sepal width).

Example 13

Moore-Penrose Generalized Inverse

The Moore-Penrose generalized inverse can be defined for any (even $m \times n$ matrix, with numeric entries) matrix **A**. This inverse is unique and it works like an inverse for solving many problems. It is denoted by **A**₋ in SYSTAT. This inverse satisfies the following four conditions:

$$\mathbf{A} \mathbf{A}_{-} \mathbf{A} = \mathbf{A},$$

$$\mathbf{A}_{-} \mathbf{A} \mathbf{A}_{-} = \mathbf{A}_{-}$$

$$\mathbf{A} \mathbf{A}_{-} \text{ and } \mathbf{A}_{-} \mathbf{A} \text{ are symmetric.}$$

SYSTAT functions available in the Matrix procedure are rich enough to yield the Moore-Penrose inverse. For any matrix A , use the following commands to obtain the Moore-Penrose inverse:

```
MAT aa = TRP(a)*a
MAT c = M(NCOL(a), 1, 1)
MAT e = I(NCOL(a))
MAT null = M(NCOL(a), 1, 0.0000001)
MAT k = EIGVAL(aa)
CALL EIGEN(d, u, aa)
MAT t = k > null
MAT tt = c - 2 # t
MAT k_ = k ## tt
MAT d_ = e # k_
MAT a_ = u*d_*TRP(u)*TRP(a)
```

$A_$ is the Moore-Penrose inverse of A .

For a consistent system of linear equations

$$Ax = b$$

A_b is a solution. Thus a solution of the system $X'X = X'Y$ is given by $(X'X)_X Y$. You may note that this is equal to X_Y .

Consider the system of equations:

$$x + y + u + v = 8$$

$$x + y - u - v = 0$$

$$x - y + u - v = 0$$

The input is:

```
MAT a = [1 1 1 1; 1 1 -1 -1; 1 -1 1 -1]
Mat b = [8; 0; 0]
```

After obtaining $A_$, the Moore-Penrose inverse of A using the above steps can be used to obtain a solution to the above system.

The input is:

```
MAT sol = a_* b
SHOW a_ sol
```


The output is:

Matrix Name: a_

	R_1	R_2	R_3
R_1	0.250	0.250	0.250
R_2	0.250	0.250	-0.250
R_3	0.250	-0.250	0.250
R_4	0.250	-0.250	-0.250

Matrix Name: sol

	C_1
R_1	2.000
R_2	2.000
R_3	2.000
R_4	2.000

There are many solutions, e.g., $(4 \ 0 \ 0 \ 4)'$ is a solution. The above solution SOL is the one that corresponds to the Moore -Penrose inverse.

References

- Birkes, D. and Dodge, Y. (1993). *Alternative methods of regression*. New York: John Wiley & Sons.
- Harville, D A. (1997). *Matrix algebra from a statistician's perspective*. New York: Springer-Verlag.
- Kutner, M. H., Nachtsheim, C. J., Neter, J., and Li, W. (2004). *Applied linear regression models*. 5th ed. New York: McGraw-Hill / Irwin.
- Rao, C.R. (1973). *Linear statistical inference and its applications*, 2nd ed. New York: John Wiley & Sons.
- Rao, C.R. and Rao, M.B. (1998). *Matrix algebra with application to statistics and econometrics*. Singapore: World Scientific.
- Schott, J.R. (1997). *Matrix analysis for statistics*. New York: John Wiley & Sons.
- Searle, S. R. (1982). *Matrix algebra useful for statistics*. New York: John Wiley & Sons.
- Seber, G.A.F. and Lee, A.J. (2003). *Linear regression analysis*. 2nd ed. Hoboken, N.J.: Wiley-Interscience.

Acronym & Abbreviation Expansions

A

ABS - absolute value
 ACF - autocorrelation function
 ACOLOR - color axes
 ACS - arccosine
 ACT - actuarial life table
 AD test - Anderson Darling test
 ADDTREE - additive trees
 ADFG - asymptotically distribution free estimate biased, Gnanian
 ADFU - asymptotically distribution free estimate unbiased
 ADJSEASON - seasonal adjustment
 AHMAX - maximum extent
 AHMIN - minimum extent
 AIC - Akaike information criterion
 AID - automatic interaction detection
 ALT - alternative
 ANCOVA - analysis of covariance
 ANG1 - deviation of angles from north in a clockwise direction
 ANG2 - deviation of angles from horizontal (for 3D models)
 ANG3 - tilt angle
 ANOVA - analysis of variance
 ANOVAHYPO - hypothesis tests in analysis of variance
 AR - autoregressive
 ARIMA - autoregressive integrated moving average
 ARL - average run length

ARMA - autoregressive moving average
 ARS - adaptive regression sampling
 ASCII - American Standard Code for Information Interchange
 ASE - asymptotic standard error
 ASN - arctane
 ATH - arc hyperbolic tangent
 ATN - arctangent
 AVERT - vertical extent
 AVG - average

B

BC - Bray-Curtis similarity measure
 BCc - Bray-Curtis corrected and accelerated
 BCJ - Beta cumulative function
 BIN - Beta density function
 BETACTR - Beta correction
 BIC - Bayesian information criterion
 BIF - Beta inverse function
 BMAP - Wenders map
 BCP - beginning of C file
 BCT - beginning of CT group
 BEND - bend
 BERT - bootstrap
 BRN - Beta random number

C

CANT - classification and regression trees
 CRSTAT - random forest classifier
 CCF - Correlation coefficient function
 CCF - cross-correlation function
 CDF - Cumulative density function
 cCDF - cumulative distribution function
 CTRVAR - coefficient for correlated variables

CFUNC - coefficients for the classification functions
 CGM - Computer graphics metafile: binary or clear text
 CHAZ - cumulative hazard
 CHISQ - Chi-square distribution
 CHOL - Cholesky decomposition
 CI - confidence interval
 CIF - Cauchy inverse function
 CIM - confidence interval of mean
 CLASS - classification
 CLSTEM - stem and leaf plot for column
 CMeans - canonical scores of group means
 CMULTIVAR - multiple string variables
 COEF - coefficients
 COL/col - column
 COLPCT - Column percentages
 CONFIG - configuration
 CONT - Contingency coefficient
 CONV - convergence
 CORAN - correspondence analysis
 CORR - correlations
 CORR1 - single correlation coefficient
 CORR2 - equality of two correlations
 COV - covariance
 Cp - process capability index
 CPL - process capability based on lower specification limit
 CPU - process capability based on upper specification limit
 Cpk-Process capability index for off-centered process
 CR - confidence region
 CRA - cost of response above UTL
 CRB - cost of response below LTL
 CRN - Cauchy random number
 CSCORE - canonical scores
 CSIZE - size of characters
 CSQ - Chi-square
 CSTATISTICS - column statistics
 CSV - comma separated values

CUSUM - cumulative sum
 CUSUM HI - Upper cumulative sum
 CUSUM LO - Lower cumulative sum
 CV - coefficient of variation
 CVI - cross validation index

D

DBF - Dbase files
 DC - deciles of risk
 DECF - Double exponential cumulative function
 DEDF - Double exponential density function
 DEIF - Double exponential inverse function
 DENFUN - density function
 dep. - dependent
 DERN - Double exponential random number
 DET - determinant
 DEVI - deviates (observed values - expected values)
 DEXP - Double exponential distribution
 df - degrees of freedom
 DF - distribution function
 DHAT - estimated distance
 DIF - data interchange format
 DIM - dimension
 DISCRIM - discriminant analysis
 DIST - distance
 DIT - dot histogram
 DOE - design of experiments
 DOS - disc operating system
 DPMO - defects per million opportunities
 DPU - defects per unit
 DTA - Stata files
 DUCF - Discrete uniform cumulative function
 DUDF - Discrete uniform density function
 DUIF - Discrete uniform inverse function
 DUNIFORM - Discrete uniform
 DURN - Discrete uniform random number
 DWLS - distance weighted least-squares

E

ECF - Exponential cumulative function

EDF - Exponential density function
 EEXP - extreme value exponential
 EIF - Exponential inverse function
 EIGEN - eigenvalues
 ELAMBDA - $\exp(\lambda)$
 EM - expectation-maximization
 EMF - Windows enhanced metafile
 ENCF - Logit normal cumulative function
 ENDF - Logit normal density function
 ENIF - Logit normal inverse function
 ENORMAL - Logit normal
 ENRN - Logit normal random number
 EOF - end-of-file
 EOG - end-of-BY group
 EPS - Encapsulated postscript
 ERN - Exponential random number
 ES - exhaustive search
 ESS - error sum of squares
 EW - extreme value Weibull
 EWMA - exponentially weighted moving average
 EXP/exp - exponential/ expected

F

FAR - false-alarm rates
 FCF - F cumulative function
 FCOLOR - color foreground
 FDF - F density function
 FIF - F inverse function
 FINV - inverse of the F cumulative
 FITC - fitting distribution: continuous
 FITD - fitting distribution: discrete
 FITDIST - fitting distributions
 Flexibeta - flexible beta
 FPLOT - function plots
 FRN - F random number
 FTD - folded trellis detector
 FTDEV - Freeman-Tukey deviate
 FULLCOND - full conditional
 FUN - function

G

GCF - Gamma cumulative function
 GCOR - groupwise correlation matrix
 GCOV - groupwise covariance matrix
 GCV - generalized cross validation
 GDF - Gamma density function
 GECF - Geometric cumulative function
 GEDF - Geometric density function
 GEIF - Geometric inverse function
 GEN - general Toeplitz structure
 GERN - Geometric random number
 GG - Greenhouse Geisser
 GIF - Gamma inverse function
 GIF - Graphics Interchange Format
 GLM - generalized linear models
 GLMHYPOT - hypothesis tests in general linear model
 GLMPOST - post hoc estimate for repeated measures in general linear model
 GLS - generalized least-squares
 GMA - geometric moving average
 GN - Gauss-Newton method
 GOCF - Gompertz cumulative function
 GODF - Gompertz density function
 GOIF - Gompertz inverse function
 GORN - Gompertz random number
 GRN - Gamma random number
 GUCF - Gumbell cumulative function
 GUDF - Gumbell density function
 GUIF - Gumbell inverse function
 GURN - Gumbell random number

H

H & L - Hosmer and Lemeshow
 HC - heteroscedasticity-consistent
 HCF - Hypergeometric cumulative function
 HDF - Hypergeometric density function
 HF - Huynh-Feldt
 HGEOMETRIC - hypergeometric
 HIF - Hypergeometric inverse function
 HIST - histogram
 HKB - Hoerl, Kennard, and Baldwin

H-L trace - Holding-Lawley trace

HR - hit-rates

HRN - Hypergeometric random number

HSD - honestly significant differences

HTERM - terms tested hierarchically

HTML - hyper text markup language

HYMH - hybrid Metropolis-Hastings

I

IF - Inverse cumulative distribution function

IGAUSSIAN - inverse Gaussian

IGCF - Inverse Gaussian cumulative function

IGDF - Inverse Gaussian density function

IGIF - Inverse Gaussian inverse function

IGRN - Inverse Gaussian random number

IIDMC - independently and identically distributed Monte Carlo

IMPSAMPI - importance sampling integration

IMPSAMPR - importance sampling ratio

I-MR - individual and moving range

Ind/indep - independent

IndMH - Independent Metropolis-Hastings

INDSCAL - individual differences scaling

INITSAMP - initial sample

INTEG FUN - integrated function

IPA - iterated principal axis

ITER - iterations

J

JACK - jackknife

JCLASS - jackknifed classification

JMP - JMP v3.2 data files

JPEG/JPG - joint photographic experts group

K

K-M - Kaplan-Meier

KNBD - kth nearest neighborhood

KRON - Kronecker product

K-S test - Kolmogorov-Smirnov test

KS1 - one sample Kolmogorov-Smirnov tests

KS2 - two sample Kolmogorov-Smirnov tests

L

LAD - least absolute deviations

LB - larger the better

LCF - Logistic cumulative function

LCHAZ - log cumulative hazard

LCL - lower control limit

LCONV - log-likelihood convergence criteria

LDF - Logistic density function

LGM - log gamma

LGST - logistic

LIF - Logistic inverse function

L-L/LL - log likelihood

LMS- least median of squares

LMSREG - least median of squares regression

LNCF - Lognormal cumulative function

LNDF - Lognormal density function

LNIF - Lognormal inverse function

LNOR/LNORMAL - lognormal

LNRN - Lognormal random number

loc - location

LOG1 - one-parameter logistic (Rasch)

LOG2 - two-parameter logistic

LOGIT - logistic regression

LOGITHYPO - hypothesis tests in logistic regression

LOGLIN - loglinear modeling

LR - likelihood ratio

LRCHI - likelihood ratio chi-square

LRDEV - likelihood ratio of deviate

LRN - Logistic random number

LS - least-squares

LSD - least significant difference

LSL - lower specification limit

LSQ - least-squares

LTAB - life tables

LTL - lower tolerance limit

LW - Lawless and Wang

M

MA - moving average

- MAD - mean absolute deviation
 MAHAL - Mahalanobis distances
 MANCOVA - multivariate analysis of covariance
 MANOVA - multivariate analysis of variance
 MANOVAHYPO - hypothesis tests in MANOVA
 MANOVAPOST - post hoc estimate for repeated measures in MANOVA
 MAR - missing at random
 MAX - maximum
 MAXSTEP - maximum number of steps
 MCAR - missing completely at random
 MCMC - Markov Chain Monte Carlo
 MDPREF - multidimensional preference
 MDS - multidimensional scaling
 MIN - minimum
 M-H- Metropolis-Hastings
 MIS - number of missing values
 MIX - mixed regression
 MIXHIER - mixed regression for data having a hierarchical structure
 MIXMULTY - mixed regression for data having a multivariate structure
 ML - Maximum Likelihood
 MLA - maximum likelihood analysis
 MLE - maximum likelihood estimate
 MML - maximum marginal likelihood
 MRC - Multiple Regression and Correlation
 MS - mean squares
 MSE - mean square error
 MSIGMA - sigma measurement
 MT - Mersenne-Twister
 MTW - MINITAB v11 data files
 MU2 - Guttman's mu2 monotonicity coefficients
 MULTIVAR - multiple variables
 MW - minimum within sum of squares deviations
 MWL - maximum Wishart likelihood

 N
 NAR - non-stationary first-order autoregressive
 NB - nominal the best
 NBB - nominal-the-best: bilateral tolerance
 NBCF - Negative binomial cumulative function
 NBD - number of active bounds on parameter values
 NBDF - Negative binomial density function
 NBIF - Negative binomial inverse function
 NBINOMIAL - Negative binomial
 NBRN - Negative binomial random number
 NBU - nominal-the-best: unilateral tolerance
 NCAT - number of categories
 NCF - Binomial cumulative function
 NCOL - number of columns
 NDF - Binomial density function
 NDMAX - maximum number of points
 NDMIN - minimum number of points
 NEM - number of EM iterations
 NEXPO - negative exponential
 NIF - Binomial inverse function
 NIPALS - Nonlinear iterative partial least Squares
 NLAG - number of lags
 NLOSS - nonlinear loss functions
 NLMODEL - nonlinear models
 NMIN - minimum count
 NMULTIVAR - multiple numeric variables
 NONLIN - nonlinear models
 NP-Number nonconforming
 NPAR - nonparametric
 NREC - non-recreationist
 NRN - Binomial random number
 NROW - number of rows
 NRP - number of apparently redundant parameters
 NSAMP - number of sub-samples
 NSPLIT - maximum number of splits
 NX - number of nodes along the x axis
 NXDIS - number of discretization points in the x (North) direction
 NY - number of nodes along the y axis
 NYDIS - number of discretization points in the y (East) direction
 NZ - number of nodes along the z axis

NZDIS - number of discretization points in the z (Depth) direction

O

Obs-observed

OBSFREQ - observed frequency

OC - operating characteristic

ODBC - open database capture and connectivity

OFREQ - outlier frequencies

OLS - ordinary least-squares

ORTHEQ- Equally Spaced Orthogonal component

ORTHUN- Unequally Spaced Orthogonal component

P

P - Proportion nonconforming

PACF - Pareto cumulative function

PACF - partial autocorrelation function

PADF - Pareto density function

PAIF - Pareto inverse function

PARAM - parameters

PARN - Pareto random number

PCA - process capability analysis

PCF - iterated principal axis factoring

PCF - Poisson cumulative function

PCNTCHANGE - percentage change

PCT - Macintosh PICT

PDF - Poisson density function

pdf - probability density function

PDL - polynomial distributed lag

PERMAP - perceptual mapping

PIF - Poisson inverse function

PLIMITS - probability limits

PLS - partial least squares

pmf - probability mass function

PMIN - minimum proportion

PNG - Portable Network Graphics

POLY - polygon

POSAC - partially ordered scalogram analysis with coordinates

P-P - probability plot

PP - process performance

Ppk - Process performance index for off-centered process

PPL - process performance based on lower specification limit

PPM - parts per million

PPU - process performance based on upper specification limit

PRE - percentage reduction error

PREFMAP - preference mapping

PRN - Poisson random number

PROB - probability

PROP1 - single proportion

PROP2 - equality of two proportions

PS - PostScript

PVAF/p.v.a.f. -- present value annuity factor

p-value - probability value

Q

QC - quality control

QMLE - quasi maximum likelihood estimate

QNTL - quantiles

QPLOT - quantile plots

Q-QPLOT - two sample quantile plot

QRD - QR decomposition

QS - quick search

QSK - quantitative symmetric similarity coefficients (or Kulczynski measure)

QUASI - Quasi-Newton method

R

R & R - repeatability and reproducibility

R chart - range chart

RADMAX - maximum horizontal direction for the search radius

RADMIN - minimum horizontal direction for the search radius

RAND - random

RANDSAMP - random sampling

RANKREG - rank regression

RBSTAT - row basic statistics
 RCF - Rayleigh cumulative function
 RDF - Rayleigh density function
 RDISCRIM - robust discriminant
 RDIST - robust distance
 RDVER - vertical direction for the search radius
 REPAR - reparametrize
 REPS - replicates
 RESID - residuals
 RIF - Rayleigh inverse function
 RJS - rejection sampling
 RMS - root mean square
 RMSEA - root mean square error of approximation
 RMSSTD - root mean square standard deviation
 ROC - receiver operating characteristic
 ROWPCT - Row percentages
 RRN - Rayleigh random number
 RS - response surface
 RSE- robust standard errors
 RSEED - random seed
 RSM- response surface methods
 RSQ - stress and squared correlation
 RSS - residual sum of squares
 RSTATISTICS - row statistics
 RTF - rich text format
 RWM-H - random walk Metropolis-Hastings
 RWSTEM - stem and leaf plot for rows

S

S chart - standard deviation control chart
 SANG1 - angle (in degrees) of the first minor axis of the search ellipsoid
 SANG2 - angle (in degrees) of the major axis of the search ellipsoid
 SANG3 - angle (in degrees) of the second minor axis of the search ellipsoid
 SAV - SPSS files
 SB - smaller the better
 sc - scale
 SC - set correlation

SCDFUNC - standardized coefficients for canonical variables
 SCF - Studentized cumulative function
 SD - standard deviations
 sd2/sas7bdat - SAS v9 files
 SDF - Studentized density function
 SE/se/S.E. - standard error
 SEK - standard error of kurtosis
 SEM - standard error of mean
 SES - standard error of skewness
 shp - shape
 SIF - Studentized inverse function
 SIMPLS - Straight-forward Implementation of Partial Least Squares
 SKMEAN - simple kriging mean
 SL - specification limit
 SMIN - minimum split value
 SPLOM - scatter plot matrix
 SQL - structured query language
 SQRT/SQR - square-root
 SRN - Studentized random number
 SRWR - sum of rank weighted residuals
 SS - sum of squares
 SSCP - sum of squares and cross products
 STA - Statistica v5 data files
 STAND - standardized deviates
 SVD - singular value decomposition
 SW - Shapiro-Wilks
 SYC/CMD - SYSTAT command Files
 SYZ/SYD/SYS - SYSTAT data files
 SYO - SYSTAT output files

T

T1 - one-sample t-test
 T2 - two-sample t-test
 TANALYZE - Taguchi design: analyze
 TCF - t cumulative function
 TCOR - total correlation
 TCOV - total covariance
 TDF -t density function
 TESTAT - Test Item Analysis

TESTATCL - classical test item analysis
 TESTATLOG - logistic item response analysis
 TETRA - tetrachoric correlations
 TGENERATE - Taguchi design: generate
 TIF - t inverse function
 TIFF - Tagged Image File Format
 TLOG - log time
 TLOSS - Taguchi's Loss Function
 TNH - hyperbolic tangent
 TOHC0 - Hypothesis Testing: Zero correlation
 TOHC1 - Hypothesis Testing: Specific correlation
 TOHC2 - Hypothesis Testing: Equality of two correlation coefficients
 TOHP1 - Hypothesis Testing: Single proportion
 TOHP2 - Hypothesis Testing: Equality of two proportions
 TOHT1 - Hypothesis Testing: One sample t-test
 TOHT2 - Hypothesis Testing: Two sample t-test
 TOHTPAIRED - Hypothesis Testing: Paired t-test
 TOHV1 - Hypothesis Testing: Single variance
 TOHV2 - Hypothesis Testing: Two variances
 TOHVN - Hypothesis Testing: Several variances
 TOHZ1 - Hypothesis Testing: One sample z-test
 TOHZ2 - Hypothesis Testing: Two sample z-test
 TOL - tolerance
 TPLOT - time series plot
 TPREDICT - Taguchi design: predict
 TRCF - Triangular cumulative function
 TRDF - Triangular density function
 TRI - triangular
 TRIF - Triangular inverse function
 TRIM - trimmed mean
 TRN - t random number
 TRP - transpose
 TRRN - Triangular random number
 TSFOURIER - Fourier decomposition of time series
 TSIV - Two-Stage Instrumental Variables
 TSLS - Two-Stage Least Squares

TSP - traveling salesman path
 TSQ chart - Hotelling's T^2 chart
 TSSMOOTH - smoothing time series
 TXT - text format

U

U chart - chart showing defects per unit
 UCF - Uniform cumulative function
 UCL - upper control limit
 UDF - Uniform density function
 UIF - Uniform inverse function
 UNCE - uncertainty coefficient
 URN - Uniform random number
 USL - upper specification limit
 UTL - upper tolerance limit

V

VAR - variance
 VIF - variance inflation factor

W

WB - Weibull
 WCF - Weibull cumulative function
 WCOR - pooled within-group correlation
 WCOV - pooled within-group covariance
 WDF - Weibull density function
 WHISKER - Box-and-Whisker plot
 WIF - Weibull inverse function
 WMF - Windows metafile
 WRN - Weibull random number

X

XCF - Chi-square cumulative function
 XDF - Chi-square density function
 XIF - Chi-square inverse function
 XLAG - separation distance between lags
 XLS - excel format
 XLTOL - tolerance for lags
 XMAX - maximum along x axis
 XMIN - minimum along x axis

X-MR chart - Individuals and moving range chart

XPT/TPT - SAS transport files

XRN - Chi-square random number

XTAB - Crosstabulations

Y

YMAX - maximum along y axis

YMIN - minimum along y axis

Z

Z1 - one-sample z-test

Z2 - two-sample z-test

ZCF - Normal cumulative function

ZDF - Normal density function

ZICF - Zipf cumulative function

ZIDF - Zipf density function

ZIF - Normal inverse function

ZIIF - Zipf inverse function

ZIRN - Zipf random number

ZMAX - maximum along z axis

ZMIN - minimum along z axis

ZRN - Normal random number

A

ABS function, 83
ACS function, 83
AND, 79
append files, 23, 26
 append cases, 26
 commands, 29
 examples, 32
ArcView files, 3, 6
ARRAY, 227
ASC\$ function, 90, 94
ASCII files, 3, 6, 19
 error messages, 208
 examples, 196, 197, 198, 199, 201, 202, 204, 206,
 209, 210, 211
 exporting, 210
 importing, 8, 194, 195, 202, 205
 missing data, 45
 troubleshooting, 207
ASN function, 83
AT2 function, 83
ATH function, 83
ATN function, 83
AVG function, 84

B

backslash, 195, 201, 202
BASIC programming, 193
 ASCII files, 194
 beginning of file, 232
 beginning of group, 232
 conditional actions, 221, 222
 data entry, 194
 dimensioning variables, 229
 end of file, 232
 end of group, 232

 examples, 196, 197, 198, 199, 201, 202, 204, 206,
 209, 210, 211, 212, 213, 214, 215, 223, 226,
 228, 230, 231, 232, 233, 234, 235, 236, 237
 line numbers, 221
 loops, 224, 227
 missing values, 221
 nested loops, 225
 print, 234
 setup, 221
 subgroups, 232
 subscripted variables, 227, 229
 transformations, 221
Benford's law, 99
Beta distribution, 102
Binomial distribution, 99
BOF, 232
BOG, 232
by groups
 see stratification

C

calculator, 126
 examples, 145
CAP\$ function, 90, 91
cases
 adding, 51
 cutting and pasting, 54
 deleting, 53
 ID, 63
 rearranging, 54
CAT\$ function, 90, 94
categorical variables, 172
Cauchy distribution, 102
center, 118
character functions, 90
 examples, 135, 137, 138, 140, 141, 142

Chi-square distribution, 102
 Cholesky decomposition, 271, 277
 CNT\$ function, 90, 92
 COD function, 86, 179
 comments
 file comments, 18
 comparing values
 examples, 189
 INC function, 180
 COMPLETE, 76
 complete cases, 76
 concatenate
 examples, 29, 30, 31, 32
 horizontally, 23
 vertically, 23, 26
 concatenating matrix, 269
 Converting Numbers from European to American
 Notation
 examples, 139
 copy, 54
 correlation matrix, 18
 COS function, 83
 covariance matrix, 18
 Cumulative distribution functions, 96, 98
 cut, 54
 CUT function, 86, 180
 cutpoints, 180
 examples, 190

D

DAT, 3, 6

data

 advanced management of, 193
 entering, 45, 64
 options, 67
 see also data entry, 45
 selection, 48
 transformations, 71, 193

data editing, 47

 adding variables, 51
 changing names, 51
 correcting values, 49
 cutting and pasting, 54

 dates and times, 50
 deleting cases, 53
 dropping variables, 53
 find values, 59
 insert variables, 56
 paste options, 55
 replace values, 59
 right-click options, 56
 undo and redo, 62

data editor, 35, 47

 active cell, 46
 case ID, 63
 navigation, 47
 opening, 3
 options, 67
 value editing area, 49
 variable statistics, 5
 variable tab, 5

data entry

 examples, 196, 197, 198, 199, 201, 202, 204, 206,
 209
 fixed-format, 202, 203
 free-format, 195
 missing values, 195
 reading multiple cases per line, 194
 reading multiple lines per case, 194
 triangular matrices, 205
 using a backslash, 195

data files

 append cases, 26
 appending, 23, 29
 commands, 28
 examples, 29, 30, 31, 32
 exporting, 19, 20, 28
 file comments, 18
 importing, 28
 merging, 23, 25, 26, 28, 29
 opening, 6
 rectangular, 36
 save options, 18
 saving, 16, 28
 saving compressed data files, 17
 spaces in filenames, 29
 spaces in paths, 29
 structure, 36
 temporary, 20, 21, 22, 28

- work files, 21
- Data Interchange Format files, 3, 6, 19
- date and time functions, 86
- date functions, 86
- dates, 40, 42, 50, 67
- dBase files, 3, 6, 19
 - importing, 9
- DBF, 3, 6
- delimited data, 195
- DIAGONAL, 206
- DIF, 3, 6
- DIM, 229
- Discrete uniform distribution, 99
- dissimilarity matrix, 18
- distribution functions, 96
 - examples, 143
- Distributions
 - univariate continuous, 101
 - univariate discrete, 99
- DOC function, 89
- Double exponential (Laplace) distribution, 102
- DOW\$ function, 90

E

- edit, 37
- editor tab
 - Data, 3
 - Variable, 37
- eigenvalues, 271
- eigenvectors, 276
- ELSE, 222
- EOF, 232
- EOG, 232
- Erlang distribution, 103
- Excel files, 3, 6, 19
 - formulas, 8
 - importing, 8
- EXP function, 83
- Exponential distribution, 103
- Exponential notation, 40
- exponentiation, 78

export

- examples, 210, 211
- export files, 16, 18
 - ASCII files, 19, 210
 - commands, 28
 - Data Interchange Format files, 19
 - dBase files, 19
 - Excel files, 19
 - JMP files, 19
 - Lotus files, 19
 - MINITAB files, 19
 - SAS files, 19, 20
 - S-PLUS files, 19
 - STATA files, 19
 - STATISTICA files, 19
- extract file, 157

F

- F distribution, 104
- files
 - see data files
- FIND, 59
- fixed-format, 202, 203
 - examples, 204
- FOR NEXT, 224
- free-format, 195
 - examples, 196, 197, 198, 199, 201, 202
- frequencies, 64
- functions, 77
 - character, 90
 - date, 86
 - distribution, 96
 - group, 86
 - mathematical, 82
 - multivariable, 84
 - time, 86

G

- gamma distribution, 104
- Geometric distribution, 100
- Gompertz distribution, 105
- group functions, 86
- grouping variables, 44, 171, 180

Gumbel distribution, 105

H

Hypergeometric distribution, 100

I

ICH function, 90, 94

IF THEN, 221

import files, 6

ASCII files, 8

commands, 28

dBase files, 9

Excel files, 8

JMP files, 6

Lotus files, 8

MINITAB files, 6

reading variable names, 7

SAS files, 9

SigmaPlot files, 8

STATA, 6

STATISTICA files, 6

StatView files, 6

INC function, 86, 180

IND function, 90, 92

INT function, 83

intervals

CUT function, 180

Inverse distribution functions, 96

Inverse Gaussian (Wald) distribution, 106

J

JMP, 3, 6

JMP files

importing, 11

K

Kronecker product, 270

L

L10 function, 83

LAB\$ function, 90

labels, 173

commands, 182

examples, 185, 187

missing codes, 175

LAG function, 83

LDISPLAY, 175

LFT\$ function, 90, 92

LGM function, 83

list cases, 151, 211

commands, 157

examples, 158, 159, 160, 164, 167, 213

picture format, 152

LOG function, 83

Logarithmic series distributions, 100

Logistic distribution, 106

Logit normal distribution, 107

Loglogistic distribution, 107

Lognormal distribution, 107

Lotus files, 6, 19

formulas, 8

importing, 8

LOW\$ function, 90, 91

LPD\$ function, 90, 94

M

matching values

INC function, 180

mathematical functions

examples, 129

matrices

correlation, 66

covariance, 66

dissimilarity, 66

input, 66

similarity, 66

SSCP, 66

types, 205

Matrix

commands, 256

decomposition, 255

examples, 258, 261, 266, 270, 272, 276, 279, 281,
282, 286, 288, 295, 297

generate, 247

operations, 252

organize workspace, 249
 overview, 243
 read, 245
 row/column, 250
 merge files, 23
 append variables, 23
 by key variables, 25
 commands, 28
 examples, 29, 30, 31
 record replication, 26
 MID\$ function, 90, 94
 MINITAB, 3, 6
 Minitab files
 importing, 10
 missing codes, 175
 missing values, 45, 195
 examples, 199
 MOD function, 83
 Moore-Penrose inverse, 297
 multcase functions, 85
 multivariable functions
 examples, 130, 131, 133, 134
 multivariate repeated measures layout
 MYSTAT files, 3

N

names, 37, 42
 NCAT functions, 173
 examples, 191
 Negative binomial distribution, 101
 Non-central chi-square distribution, 108
 Non-central F distribution, 108
 Non-central t distribution, 109
 Normal distribution, 108
 NOT, 79
 NOW\$ function, 87
 numeric variables, 39

O

ODBC, 11
 data sources, 12
 databases, 12, 13

SQL, 15
 tables, 13
 variable selection, 14
 operators, 77
 arithmetic, 78
 logical, 79
 order of evaluation, 81
 relational, 78
 options
 data, 67
 ordering categories, 175
 commands, 183

P

p value, 145
 Pareto distribution, 109
 Poisson distribution, 101
 PRINT, 211, 213, 214
 properties, 42, 43
 PUT\$ function, 90, 94

Q

QR decomposition, 277

R

random deviates, 115
 random number generator, 116
 Mersenne-Twister, 116
 Wichmann-Hill, 116
 Random variate functions, 97
 ranks, 117
 commands, 127
 Rayleigh distribution, 110
 recode, 177
 commands, 183
 examples, 187
 recoding variables
 COD function, 179
 examples, 188, 189
 rectangular files, 18, 37
 repeated measures, 121
 reshape data, 121

RGT\$ function, 90, 92

RPD\$ function, 90, 94

S

SAS files, 3, 6, 19

data files, 9

dates, 20

exporting, 20

importing, 9

numeric variables, 9, 20

PROC COPY, 9

PROC CPORT, 9

string variables, 9, 20

times, 20

transport files, 9

variable names, 20

variable values, 9

SAV, 7

save cases, 211

examples, 213, 214

saving, 173

category information, 173

compressed data files, 17

data files, 16

value labels, 175

SD2, 3, 7

select cases, 156, 211

commands, 157

examples, 167, 212, 213, 214

selecting cells, 48

SGN function, 83

SHP, 3, 7

SigmaPlot files, 7, 8

significance levels, 115

similarity matrix, 18

SIN function, 83

singular value decomposition, 278

Smallest extreme value distribution, 110

SND\$ function, 90, 95

sort cases, 154

commands, 157

examples, 165, 166

sort order, 155

split plot designs

S-PLUS, 3

SPSS files, 3, 6

SQL

SELECT statement, 15

SQR function, 83

SQZ\$ function, 90, 92

SSCP matrix, 18

stacking variables, 124

examples, 147

standardized scores, 119

commands, 127

examples, 144

STATA, 3

STATISTICA, 3, 6

STATISTICS files

importing, 10

StatView, 3, 6

STR\$ function, 87, 88, 90, 93

stratification, 171, 180, 181

commands, 182

string variables, 39

structured query language

case selection, 15

Studentized maximum modulus distribution, 111

Studentized range distribution, 111

SUB\$ function, 90, 94

subscripted variables, 227, 229

sweep function

evaluating a quadratic form, 291

inverting matrices, 289

multiple correlation coefficient, 294

solving the system, 292

stationary distribution for Markov chain, 292

SYD, 3, 6

SYS, 3, 6

SYSTAT files, 3, 6

SYZ, 17

T

t distribution, 112

TAN function, 83
templates
 reserved tokens, 22
temporary data files, 20, 21, 22, 28
time functions, 86
times, 42, 50, 67
TNH function, 83
transformations, 71
 commands, 127
 conditional, 74
 shortcuts, 76
 unconditional, 73
transport files, 9, 19
transpose files, 121
 commands, 28
Triangular distribution, 112
triangular matrices, 205
 examples, 206
trimming cases, 120
 examples, 146
TXT, 3, 6
TYPE, 205
types, 42

U

Uniform distribution, 113
unwrap, 123
UPR\$ function, 90, 91

V

VAL function, 87, 88, 90, 93
variable
 editor, 37
 properties, 37, 42, 43
variables
 adding, 51
 built-in, 76
 changing names, 51
 comments, 44
 cutting and pasting, 54
 default names, 46
 defining, 37, 42, 43

dropping, 53
frequencies, 64
grouping, 44
insert, 56
missing data, 45
reading names, 7
rearranging, 54
subscripted variables, 44
types, 39

W

Weibull distribution, 114
weighting cases, 125
WHILE...ENDWHILE, 225
WK1, 3, 6, 8
WKS, 3, 6, 8
wrap, 123

X

XLS, 3, 6, 8
XPT, 3, 6

Z

z scores, 119
Zipf distribution, 101